United Business Media

FEATURE: 2009 GAME ENGINE SHOWDOWN

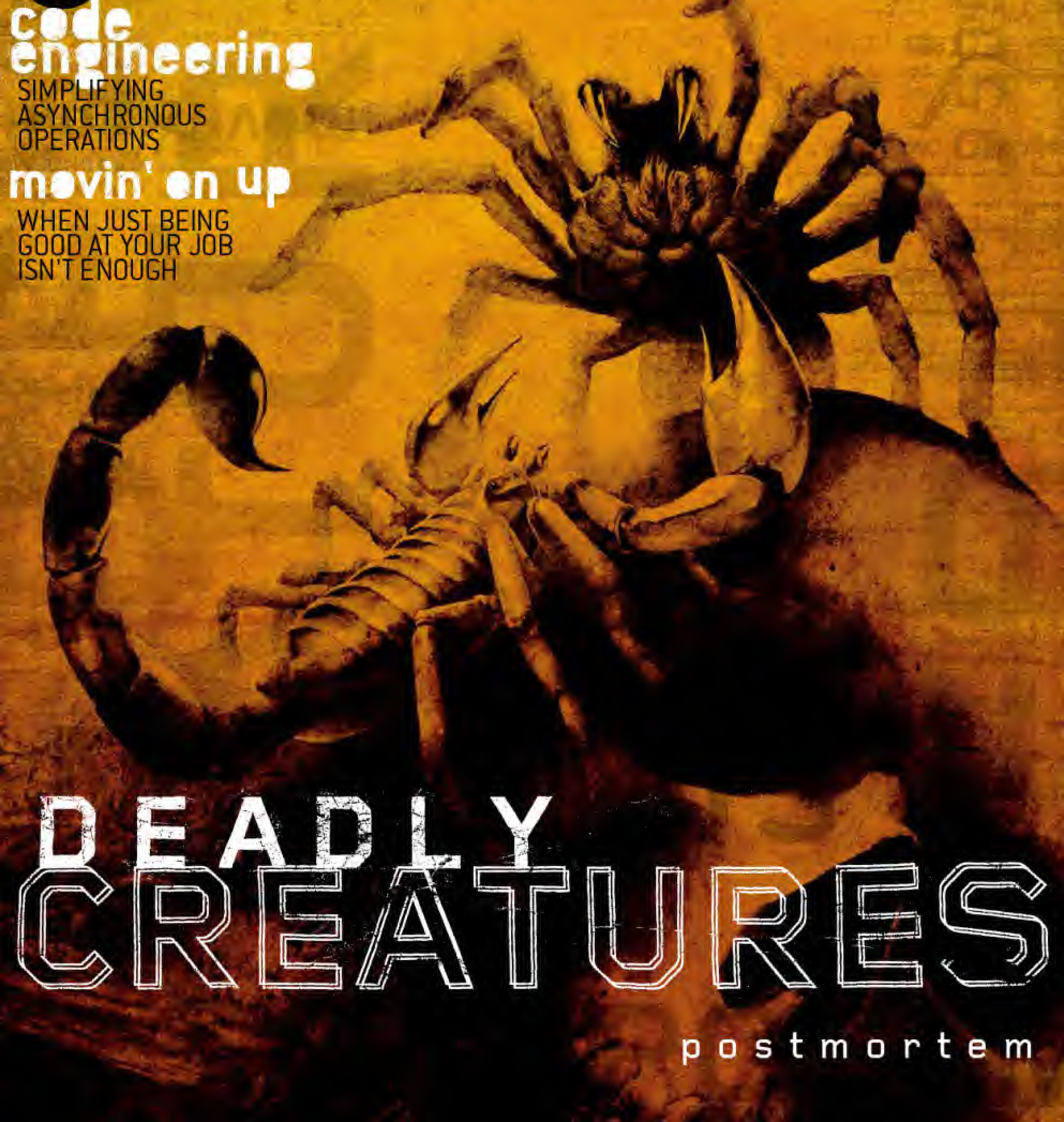
VOL16NO5MAY2009

game developer

THE LEADING GAME INDUSTRY MAGAZINE

code engineering
SIMPLIFYING ASYNCHRONOUS OPERATIONS

movin' on up
WHEN JUST BEING GOOD AT YOUR JOB ISN'T ENOUGH



DEADLY CREATURES
postmortem

EYETRONICS 3D

3D SCANNING SOLUTIONS

CUSTOMERS INCLUDE:

ROCKSTAR
ELECTRONIC ARTS
UBISOFT
SONY
2K INTERACTIVE
MIDWAY, THQ, etc...

EYETRONICS
www.eyetronics.com
800.205.9808.EU +32.16.298309

CONTENTS.0509
VOLUME 16 NUMBER 5

COVER SOURCE ART: DEADLY CREATURES ART TEAM

DON'T HATE THE GAME

RUMINATIONS ON DEVELOPER GAME-PLAYING MALAISE

IN A PREVIOUS COLUMN (November, 2008), I mentioned that it would be beneficial for developers to look outside games for inspiration. This is something I believe strongly, but on top of that, how many of you out there actually have the time to even play games, let alone consume other media?

It seems that nine times out of ten, when I ask a working developer what games he's played recently, he'll honestly admit he doesn't have the time to play any games but his own. Those that say they have played contemporary titles, if pressed, often admit only a cursory familiarity with the recent games they've tried. Some actually seem to be proud of the fact they don't actively play games but their own. This is a world-wide phenomenon, and not a particularly awesome one.

FIRST, A COUNTER-EXAMPLE

» I recently heard a story from a friend who used to be a producer on the publishing side. He played a lot of games on his own time, and talked about it vocally with others in the office, as watercooler discussion. Over time, he became known as a guy who plays a lot of new releases—even by his higher ups. His bosses would start to come around, asking for opinions on competing titles, and he'd be able to give solid answers. As a result of being known as the guy who knows about games, he got promoted to an executive-level position dealing with third parties.

Now, this guy is competent and intelligent, so those are contributing factors to his rise, but even to hear him tell it, his having played contemporary games was so unique and valuable that it warranted a promotion.

KNOWING ME, KNOWING YOU, A-HA

» Games are, like most entertainment media, very strongly influenced by past successes. If you make an FPS, you're not just referencing the CALL OF DUTY series, you're riding on the shoulders of DOOM, WOLFENSTEIN 3D, and even AD&D: TREASURE OF TARMIN for the Intellivision. No product exists in a vacuum, and in this world of iterative improvements, those games which are made with an awareness of the past are less likely to repeat mistakes, and more likely to push forward.

I've heard some folks say they don't want to be accused of being influenced by other games, but theoretically if your design is solid and well-implemented, it should stand on its own. And as I mentioned, what game isn't influenced by a host of others? Leads at the very least should be paying attention to the work going on in other studios, or should be playing those studios' games. After all, what director doesn't watch movies, and what novelist doesn't read books? Certainly only the outliers.

THE USUAL CAVEAT

» Time is every game developer's nemesis. A 60 hour work-week is not unusual, and if you've got a family, how can you justify playing games at home (other than perhaps with your child or spouse)? There is a lot of institutional pressure keeping developers from playing competing products, and some of them may not be solvable in the short term. If people could stop working 60 hour weeks, they would. In the film industry, the whole team works ridiculous hours for the duration of the project, but they are compensated well enough that they can actually take a bit of a break. A potential solution might be to have mandatory scheduled playtime for leads and key creatives during work hours. Of course, when the pedal's to the metal, that looks like an attractive cut, but building it into company culture would likely be beneficial.

I personally have been trying out the first hour of any game I get, rather than filing it away for the day when I'll "really have time to sit down with it." You can glean a lot from that first hour, and if it's engaging, you might go for another. Reading reviews and gathering popular opinion on a title just isn't enough. No judgment is more sound than our own, yes? In some cases it may be impossible to work games into your life more than they already are—but it seems like something worth doing.

NOUVELLE VAGUE

» I'd also like to point out something you may have already noticed—our redesign, the first this magazine has had since 2003. We began the process in the April issue, and believe now that we've taken it about 90 percent of the way to its destination. Praises go to our art director Joseph Mitch and production editor Jeffrey Fleming—the whole team collaborated to try to smooth out some of the magazine's rougher edges (though naturally Joe did most of the actual work).

We feel the magazine is now straddling a fine line between the hardcore programmer-oriented whitepapers and the more broadly accessible articles like What Went Wrong? (December, 2008) and Dirty Coding Tricks (March, 2009). With any luck, our current redesign reflects our interstitial position between both spaces, and will be a welcome change. If you love it, hate it, or have any other comments regarding the recent direction of the magazine, drop a line to bsheffield@gdmag.com. We do make this magazine for you, and we want to make the best use of our pages as is possible.

—Brandon Sheffield

gamedeveloper
www.gdmag.com

Think Services, 600 Harrison St., 6th Fl.,
San Francisco, CA 94107
t: 415.947.6000 f: 415.947.6090

SUBSCRIPTION SERVICES

FOR INFORMATION, ORDER QUESTIONS, AND ADDRESS CHANGES
t: 800.250.2429 f: 847.763.9606
e: *gamedeveloper@halldata.com*

EDITORIAL

PUBLISHER
Simon Carless | *scarless@gdmag.com*
EDITOR-IN-CHIEF
Brandon Sheffield | *bsheffield@gdmag.com*
PRODUCTION EDITOR
Jeffrey Fleming | *jfleming@gdmag.com*
ART DIRECTOR
Joseph Mitch | *jmitch@gdmag.com*
CONTRIBUTING EDITORS
Jesse Harlin | *jharlin@gdmag.com*
Steve Theodore | *stheodore@gdmag.com*
Noel Llopis | *nllopis@gdmag.com*
Soren Johnson | *sjohnson@gdmag.com*
Damion Schubert | *dschubert@gdmag.com*
COPY EDITING
Jill Duffy | *jduffy@gdmag.com*
ADVISORY BOARD
Hal Barwood Designer-at-Large
Mick West Independent
Brad Bulkley Neversoft
Clinton Keith High Moon Studios
Ryan Lesser Harmonix
Mark DeLoura Independent

ADVERTISING SALES

GLOBAL SALES DIRECTOR
Aaron Murawski e: *amurawski@think-services.com*
t: 415.947.6227
MEDIA ACCOUNT MANAGER
John Malik Watson e: *jmwatson@think-services.com*
t: 415.947.6224
GLOBAL ACCOUNT MANAGER, EDUCATION AND RECRUITMENT
Gina Gross e: *ggross@think-services.com*
t: 415.947.6241
COORDINATOR, EDUCATION AND RECRUITMENT
Rafael Vallin e: *rvallin@think-services.com*
t: 415.947.6223

ADVERTISING PRODUCTION

PRODUCTION MANAGER
Robert Steigleider e: *rsteigleider@ubm-us.com*

REPRINTS

WRIGHT'S REPRINTS
Ryan Pratt e: *rpratt@wrightsreprints.com*
t: 877.652.5295

THINK SERVICES

CEO THINK SERVICES Philip Chapnick
GROUP DIRECTOR Kathy Schoback
CREATIVE DIRECTOR Cliff Scorso

AUDIENCE DEVELOPMENT

GROUP DIRECTOR Kathy Henry
e: *khenry@techinsights.com*
DIRECTOR Kristi Cunningham
e: *kcunningham@techinsights.com*
LIST RENTAL Merit Direct LLC t: 914.368.1000

MARKETING

SERVICES MARKETING COORDINATOR Laura Robison
e: *lrobison@think-services.com*

UBM TECHNOLOGY MANAGEMENT

CHIEF EXECUTIVE OFFICER David Levin
CHIEF OPERATING OFFICER Scott Mozarsky
CHIEF FINANCIAL OFFICER David Wein
CHIEF INFORMATION OFFICER Kevin Prinz
CORPORATE SENIOR VP SALES Anne Marie Miller
SENIOR VP, STRATEGIC DEV. AND BUSINESS ADMIN. Pat Nohilly
SENIOR VP, MANUFACTURING Marie Myers

United Business Media
WWW.CMPGAME.COM

TRINIGY
www.trinigy.net

EVALUATING 3D GAME ENGINES?

Trinigy's Vision Game Engine offers:

- ✓ Multi Platform: PC, XBOX360, PS3, Wii
- ✓ Proven Technology: More than 100 titles
- ✓ Mature Real-Time Tools: Stable and extensible
- ✓ Rapid Game Development: Edit, test, iterate
- ✓ Industry Standard Middleware Integrations
- ✓ High Performance: Multi-core architecture
- ✓ Dedicated Middleware Provider
- ✓ First-class Support

✓ Check us out

Wii WiiWare PLAYSTATION 3 XBOX 360 Games for Windows

OVERHEARD AT GDC 09

GREG ZESCHUK AND RAY MUZYKA

BioWare heads and Electronic Arts VPs on their transition from creative to more management-oriented positions.

Greg Zeschuk:

"In the early days, we were right in there as the producers on the game, working on stuff. Now, I think it's still a very rewarding job, because we get to work across all the products. You look at things from a portfolio perspective in terms of all the games you want to work on, and you actually tend to work on the teams themselves—sort of, "How do we help our teams be successful? How do we help them be among the best?"

Ray Muzyka:

"We coach or mentor them.

[Over a decade ago] we were directly producing games ourselves. Now we have great exec producers and project directors, leads that are responsible and accountable and delivering on all fronts for our games.

From our perspective, now we get to play the games and just really enjoy them as consumers, and offer feedback from that perspective, at the early start of the ideation phase. We say, "What's our audience excited by? What's our aspirational fantasy? What are we solving for here? What goals are we setting up for this project?" Then the team goes and they work on that. We play it throughout the process and give feedback whether the team's meeting the goals that they set out at the start of the process.

Left: Muzyka. Right: Zeschuk.

EMIL PAGLIARULO

Lead designer and writer of FALLOUT 3, on balancing main quests and sub-quests.

"The main quest is where we like to tell our story. But all of the side quests that we do aren't really connected to the main quest in any way; most of the time, they're not even connected to each other. They just fill in the world, and they're just out there for you to find.

That's one of the benefits—the player can jump between one and the other at any time. It's interesting how we concentrate our time, because we spent a lot of time working on the story for the main quest and its polish and stuff,

but at the same time, we have almost two games there to make what the player's experiencing. It's impossible for us really to track what that experience for the player is going to be. Do they do two quests of the main quest, do thirty hours of the side quests, then come back and finish the main quest? Do they just beeline through the main quest? We find that most people don't do that.

When we design the game, we tend to structure it so that the player traverses the map. We'll actually move the quests to achieve that: "Let's put this over here because when they're on the main quest, they're going to run into this location." They're two separate things, but they're symbiotic, too.

STEVE MERETZKY

Classic adventure game maker and VP of game design at Playdom, on the plight of game writers:

"I'm a big advocate that writers shouldn't just be someone who you bring on two months before the game ships as a "Oh, the game is almost done; add some writing." It's much better for them to come early on so that, for one thing, they can be a lot more familiar with the game and do a lot better job when it is time to do the writing, so that they can do the writing in stages and sort of provide almost a first draft of the writing. And that will make the game much more playable for everyone who is playing early builds of the game. And then polish those drafts,

as the game gets closer to release. The writer, by coming early, is then in a position to make a lot more suggestions about the design of the game where they see that will aid the writing or that will avoid hurting the writing.

I think for the people who do only writing, more often than not, it's a pretty frustrating experience because they don't feel like they have enough of a creative role. They feel like they are just sort of being treated as a compartmentalized craftsman, and they don't feel like they were brought in early enough.

HIROSHI MATSUYAMA

CEO of Cyber Connect2, on his early difficulties with work/life balance:

"For three years I basically lived at work. I paid rent on an apartment that was empty; that I never lived at. Because of the fact that I was an amateur entering into a world where there were other professionals working, I had to work three times as hard as everybody else—and that was a reality that I couldn't escape: no matter how hard I pushed myself, there just wasn't enough time, because I didn't know the industry. And so, I had to stay at work three times as long as everybody else.

[After everyone] would go home, I was alone, by myself, working on things. I would look at what I had made, and wasn't satisfied, so I had to fix what I had done during the day. And I was looking

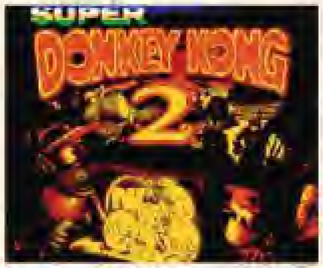
at other people's stuff, and wasn't satisfied with what they had done either! So I was messing around and fixing the things that they had been making during the day. So I was there all night long. But, what I would do is, I would stick my head under the kitchenette's warm water spout, and wash my hair there, and strip down and take a towel and take a sponge bath. And since there's nobody there, I can completely just take it all off. But that's just how I lived, day after day! And I lost a lot of weight.

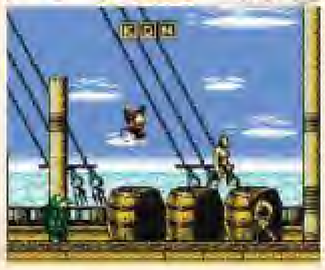
PIRATE PORTS

UNUSUAL CONVERSIONS TO THE NES

The latest and greatest consoles don't always make it to the developing world. In China, the gaming scene of the 1990s was dominated by the Famicom—or at least the Famiclone. Still, the country's booming magazine industry kept a stream of teasers flowing in from Japan that left gamers eager to play the latest and greatest. Enter the pirates.

The fabricators of unlicensed Famicom cartridges—usually IP pirates, but occasionally developers—came up with a slew of really creative ways to backport titles from fourth and fifth-generation systems to the "Red and White Machine." —*Derrick Sobodash*





SUPER DONKEY KONG 2

CHINESE TITLE: Chaoji Da Jingang

PUBLISHER: Ka Sheng

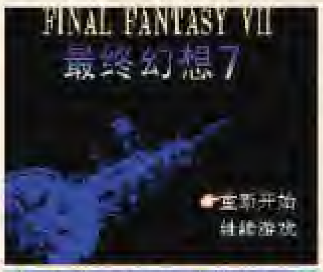
PRICE: Unknown

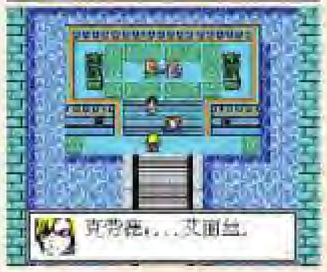
Rare took a big financial risk when it invested in SGI workstations to make DONKEY KONG COUNTRY: the Super Nintendo's (SNES) third bestselling game. DONKEY KONG brought cutting-edge 3D technology to a comparatively old system and paved the way for similar games like KILLER INSTINCT. Ka Sheng brings its sequel, DIDDY'S KONG QUEST, back to 8 bits.

THE GOOD: Ka Sheng's game plays as solid as the original, though with smaller sprites relative to the screen size.

THE BAD: The rich, CGI-rendered characters appear here in three colors. They still look good, but lose one of the series' key selling points. The enemies also respawn too soon, making backtracking dangerous.

THE UGLY: Only four stages? This engine is too good for such a short game.





FINAL FANTASY VII

CHINESE TITLE: Zuizhong Huangxiang VII

PUBLISHER: ShenZhen Nanjing Technology

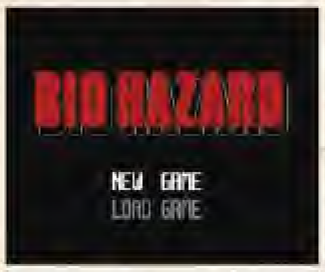
PRICE: 35 yuan ($5.12 USD)

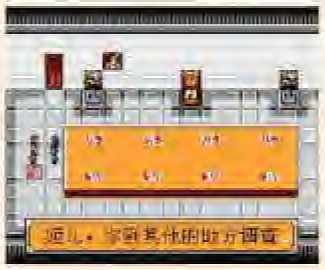
Square brought in millions of converts to the RPG genre with FINAL FANTASY VII. This FINAL FANTASY installment was praised for its soundtrack, narrative and "beautiful cut-scenes." But what happens when those scenes vanish? ShenZhen Nanjiing answers that on its custom Famicom board.

THE GOOD: ShenZhen crushed 3 CD-ROMs into a 16-megabit cartridge. The story survived unscathed—if you can live without Yuffie and Vincent. The game even has a Materia system, and the death of Aeris!

THE BAD: Zero originality. The script was lifted from the Chinese translation of FINAL FANTASY VII. Sprites and backgrounds were pillaged from FINAL FANTASY III. Character portraits were pilfered from SUPER ROBOT WARS. Did ASCII make a *Final Fantasy Maker* 1990?

THE UGLY: The soundtrack is terrible 8-bar loops of early FINAL FANTASY music hacked to fit Nanjing's driver. Battles are unbearably slow and unbalanced, and surviving to the first item shop depends entirely on luck.





BIOHAZARD/ RESIDENT EVIL

CHINESE TITLE: Sheng Hua Weiji

PUBLISHER: Waixing Computer Science and Technology

PRICE: 25 yuan ($3.66 USD)

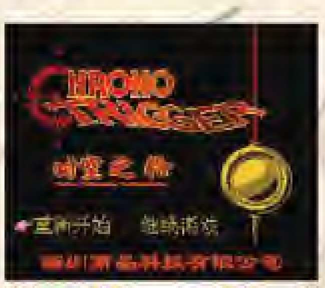
URL: www.waixing.com.cn

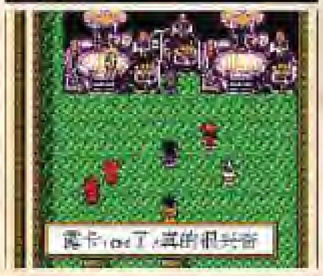
Waixing has a dirty name in the industry, but turned out a few gems in its early days. In this RESIDENT EVIL clone, the player guides Jill Valentine through a 2D mansion during the first T-Virus outbreak in Raccoon City. This Famicom remake comes out sufficiently scary.

THE GOOD: The zombies are fast enough to offset any loss in tension that came with the switch to an overhead view. Major gameplay elements like mixing items and hunting for ammo were preserved—for better or for worse.

THE BAD: While Waixing might have made the map sprites, it stole the zombies and battle system from RESIDENT EVIL GAIDEN. That's not necessarily bad, but the first-person shooter style battle engine, based on a bouncing-line rhythm game, would be a lot cooler if it used the NES Zapper.

THE UGLY: Repetitive, rumbling music. Ambience is key to survival horror, and that's where Waixing loses out. For a Famicom game with suitably scary music check out Capcom's SWEET HOME.





CHRONO TRIGGER

CHINESE TITLE: Chao Shikong Zhi Lun

PUBLISHER: ShenZhen Nanjing Technology

PRICE: 35 yuan

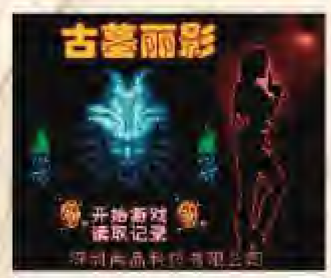
URL: www.sznanjing.com

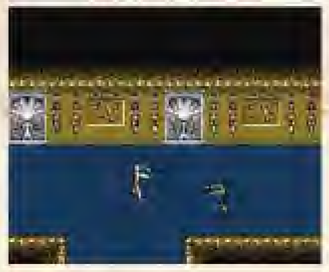
CHRONO TRIGGER pushed the limits of the Super Nintendo's base hardware with its rich visuals and moving music. But just as important to the experience was its story, something a Famicom clone cannot possibly get wrong. Right? The Black Wind howls for whoever dares to play this disaster.

THE GOOD: Nanjing tried to copy three of the game's songs, but the beautiful melodies of Yasunori Mitsuda end up an ear-shattering mess with scratchy drums.

THE BAD: Guardia 1000 A.D. has a suspicious number of PCs and Pokéballs. The game is clearly hacked over Nanjing's pirate POKÉMON—Crono's first fight is against a pack of wild Rhyhorn. Every plot point is changed, and the game ends when you meet Magus.

THE UGLY: The active battle system, laggy scroll speed, and poor list organization combine in a fatal cocktail. Clearly no testing was done before this game shipped, since it is unbeatable without some impressive RAM hacking.





TOMB RAIDER

CHINESE TITLE: Gumu Li Ying

PUBLISHER: ShenZhen Nanjing Technology

PRICE: 15–30 yuan

URL: www.sznanjing.com

Eidos/Core Design came out of nowhere with TOMB RAIDER, one of the first fully 3D adventure games on the PlayStation (and Saturn) and a key seller in the systems' early libraries. Critics were as taken by the idea of an adventure game with a heroine as they were by her boxy curves. But all the mid-1990s girl power in the world can't make this 3D adventure work on Famicom.

THE GOOD: Nanjing wrote a new story. In this game, Lara has to locate a missing map whose finders have all died mysterious deaths. Is it a curse or something else?

THE BAD: Lara runs like she's weighted down by two really heavy objects: her guns. Movement slows to a crawl when the pistols come out, and those one-pixel bullets would be more at home on an Atari 2600.

THE UGLY: Hope you like the title screen music, because that is the entire soundtrack. Damage calculation is stupid: you have to bury six slugs in a spider before it dies. Once you realize bringing up the menu resets all enemy positions, you'll be abusing that glitch to avoid playing this game.

www.seapine.com/gamescm
Satisfy your quality obsession.

1 + 1 = infinite possibilities

© 2009 Seapine Software, Inc. All rights reserved.

Get More Change Management with Seapine Integrated SCCM

TestTrack Pro + Surround SCM = infinite SCCM possibilities. Seapine's integrated software change and configuration management (SCCM) tools do much more than competing tools, and at a much lower price point. Start with TestTrack Pro for change management and add Surround SCM for configuration management– two award-winning tools that together give you the best integrated SCCM solution on the market.

- Link issues, change requests, and other work items with source code changes.
- Manage simple or complex change processes with flexible branching and labeling.
- Coordinate distributed development with RSS feeds, email conversation tracking, caching proxy servers, change notifications, 3-way diff/merge, and other collaboration features.
- Enforce and automate processes with incredibly flexible work item and file-level workflows.

Built on industry-standard RDBMSs, Seapine's SCCM tools are more scalable, give you more workflow options, and provide more security and traceability than competing solutions.

Get more, do more with Seapine tools. Visit **www.seapine.com/gamescm**.

Seapine Software™

QA Wizard® Pro Automated Testing

Seapine CM® Change Management

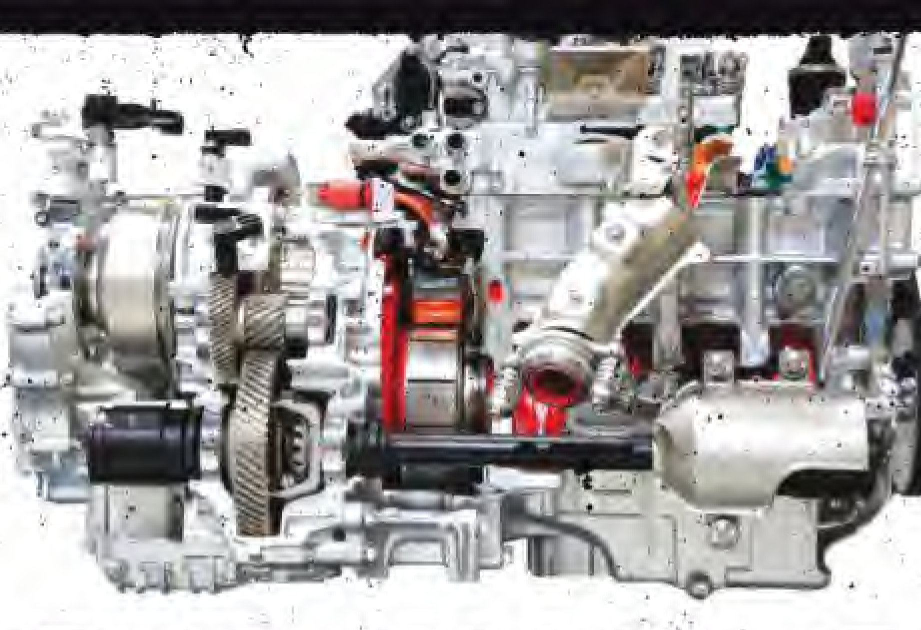
Surround SCM® Configuration Management

TestTrack® Studio Test Planning & Tracking

TestTrack® TCM Test Case Management

TestTrack® Pro Issue Management

M A R K   D E L O U R A



game engine SHOWDOWN

what game engines are studios using, and why?

GAME ENGINES HAVE BECOME BIG BUSINESS. IN THE PAST YEAR, THERE HAS BEEN A DRAMATIC increase in the number of companies selling their game engines as middleware. Engines are now available for every game category, from casual to mobile to social to MMO.

From the developer's perspective, questions about game engines have shifted from, "Why would I use that?" to "Does it make sense for my game, genre, category, budget, and timeline?"

To find out the general perceptions of licensed game engines, as well as what developers want from them, I surveyed around 100 game industry technology and production executives (roughly 60/40), primarily in the core games space in North America. For the purpose of this article, "core" games refer to high-budget games for PlayStation 3, Xbox 360, and PC—or AAA titles. (The casual and mobile markets have different engine demands, which sometimes overlap with the core, but the information contained here should not be taken as directly applicable to those markets.)

I wanted to find out from the decision makers of engine purchases whether they're using a licensed game engine, what they look for, and how much they are willing to pay.

I first asked survey respondents, "For what platforms are you currently developing titles?" (see Table 1, Pg 8). Respondents were able to select more than one platform, and the top three were Xbox 360 (70.6%), PlayStation 3 (66.2%), and PC (55.9%).

The budgets for these titles fell largely into two groups: less than $4 million (42.2%) and more than $16 million (42.4%).

GAME ENGINE AWARENESS AND USE

>> One of the most important pieces of the business puzzle for game engine companies is getting their product in front of the right people. Executives can't purchase an engine if they don't know it exists, so one issue this survey considered was awareness.

The game engines that our respondents were most aware of were not a huge surprise (see Table 2 Pg 9). The top five were: Unreal (93.4%), CryENGINE (92.1%), Torque (90.8%), Gamebryo (85.5%), and Source (82.9%). Comparatively, both the technology specialists and producers surveyed had roughly the same level of awareness of the engines.

However, when asked which of these game engines they've actually used, the picture becomes more interesting (see Table 3 Pg 9). A little more than 60% had used Unreal, 23.7% had used Torque, and 22.4% had used Gamebryo. Despite its high awareness rating, only 9.2% had actually used CryENGINE.

PERCEIVED USEFULNESS

>> Beyond simple awareness, how useful do executives think the various game engines would be for their current projects? Respondents were asked to rate the perceived usefulness of each engine on a scale of 1 to 5 (5 being highest), or they could opt out if they didn't know. Table 4 (Pg 9) shows the results.

Something that pops out of this data when combined with the previous questions is the belief that CryENGINE is the second most useful game engine middleware available, even though it is one of the least used! These ratings point to a perception of a game engine's capabilities, rather than actual user experience.

Also notable is the sudden emergence of idTech and Infernal in the top five, in spite of not being widely used. Does it reflect a belief that game engines made by companies that are using them for their own games must be the most useful? Or, bearing in mind that the question asked respondents to think about their own current projects, could it be that these engines are being used for games of similar genres or on the same platform, and thus seem to be highly relevant and useful?

Certainly some of the other game engines pride themselves on being more broadly useful across a variety of genres, but the data collected here may suggest a belief that genre specialization is valuable.

A third point immediately noticeable is that, due to the respondents largely being focused in the core space, Unity and Torque did not fare as well. This finding should not be taken as indicative of these engines' actual usefulness, but rather a reflection of the people who were polled for this survey and the kinds of projects they have in development.

SERVICES OFFERED

>> The practices by engine manufacturers vary significantly, in terms of support, source availability, delivery, and sticking to a roadmap. I asked technologists about a few important practices to see which would stand out. Table 5 (Pg 11) shows

the ones they felt were the most important.

Having access to source code and easy integration with other middleware libraries were most important—even more so than details of the engine itself, such as having the ability to modify memory allocators.

Some of the engine providers give continual access to new engine builds, and this is valuable indeed, but seems less valuable than the current working feature set. The fact that having a clear engine roadmap is judged as being as valuable as ongoing access seems to imply a desire to lock in on an engine version and not worry about upgrades during the development process. (This is mirrored in the way that many studios lock in on a version of their third-party tools during the span of a game's development.)

One respondent made a very important comment about engine development roadmaps: "I would not purchase nor use any middleware where I am dependent on a future feature. Tying my studio to the whims of development of a third party is not good business." I think that says it all.

WHY BUY?

>> Although a lot of game development teams are using licensed game engines, many still use their own. The technologists and producers surveyed for this article agreed on three primary reasons for using a third-party engine.

First, artists and designers can begin working immediately. Without engine tools and technology available, it's difficult to gauge the limitations of the engine and the unique features available to the game designers and artists. Second, engineers can focus on game-specific code. Why spend programmer-hours writing memory allocation code when the team can instead work on tweaking the physics to suit the particular style of the game being created? Third, and tightly related to the two previous points, many engine users believe that using a game engine allows them to shorten the development cycle overall, reducing the amount of work needed on the part of the engineering team and allowing the content creators to start working earlier in the development process.

Let's look at each of these points in more detail.

PROTOTYPING AND ITERATION

>> Why is early ramp-up and quick iteration time so important? Is it to save costs by allowing a smaller team to develop more rapidly? Not necessarily. In fact, 69% of producers surveyed said that they expect using a game engine to have no impact on staff size. Instead, it allows them to focus more on game-specific details. They mentioned getting to "focus on enhancements and differentiating features."

Having early-stage development tools is useful in two major ways. First, having tools early allows artists and designers to begin working immediately on content that may actually wind up in the game. Without such a toolset, they are limited to more conceptual, pre-production, pen-and-paper work, which, while valuable, probably won't end up in the final product.

Second, developing prototypes is an important process that helps establish the game's unique tone and direction before gearing up for full production. Finding the fun in a game concept in the early stages of a project makes it much easier to pitch and get funded, and highlights the game development challenges that will have to be solved during the length of the project. Without the ability to rapidly create prototypes to determine the features that will be most important to the game, the early part of the title's production will be much less focused.

One myth I'd like to dispel is that game developers have solved these challenges already. Recently a developer wrote me and reasoned, "over the past few years, the industry has discussed prototyping, and now it is a solved problem." Tell that to the independent studio that starts out tool-less. We all may now agree that prototyping is important, and it may well be the case that established studios and publishers that share technology among their teams have solutions in place, but as far as tools for small studios go, prototyping solutions are still not widely available outside a full game engine purchase.

Table 6 (Pg 12) shows what tools developers are using now for prototyping. In addition to the tools shown in Table 6, other respondents noted that they've used Epic's Unreal Engine, Emergent's Gamebryo, Vicious Engine, Google SketchUp, and Adobe Photoshop.

It's hard to replace the good old pencil and paper for ease of use, but the relatively low response on the use of XNA was surprising, since several teams are known to have used it for prototyping with good success. C/C++ is of course an obvious choice for developing any playable prototypes, but Flash, Lua, and XNA have been great productivity multipliers for many teams.

Rapid iteration is also considered a valuable factor in creating high quality games. The ability for any member of the team to test out new content in the game (whether it be art, audio, or code), and do it easily and rapidly, allows for more experimentation and fine-tuning. If it takes 15 minutes for an artist to see her work in the game engine, how likely is she to

engines referenced

Crytek's CryENGINE 3
www.crytek.com/technology/cryengine-3/specifications

Digital Extremes' Evolution Engine
www.digitalextremes.com/tech

Emergent Game Technologies' Gamebryo
www.emergent.net

Epic Games' Unreal Engine
www.unrealtechnology.com

Garage Games' Torque
www.garagegames.com

Id Software's idTech
www.idsoftware.com/business/technology

Terminal Reality's Infernal Engine
www.infernalengine.com

Trinigy's Vision
www.trinigy.net

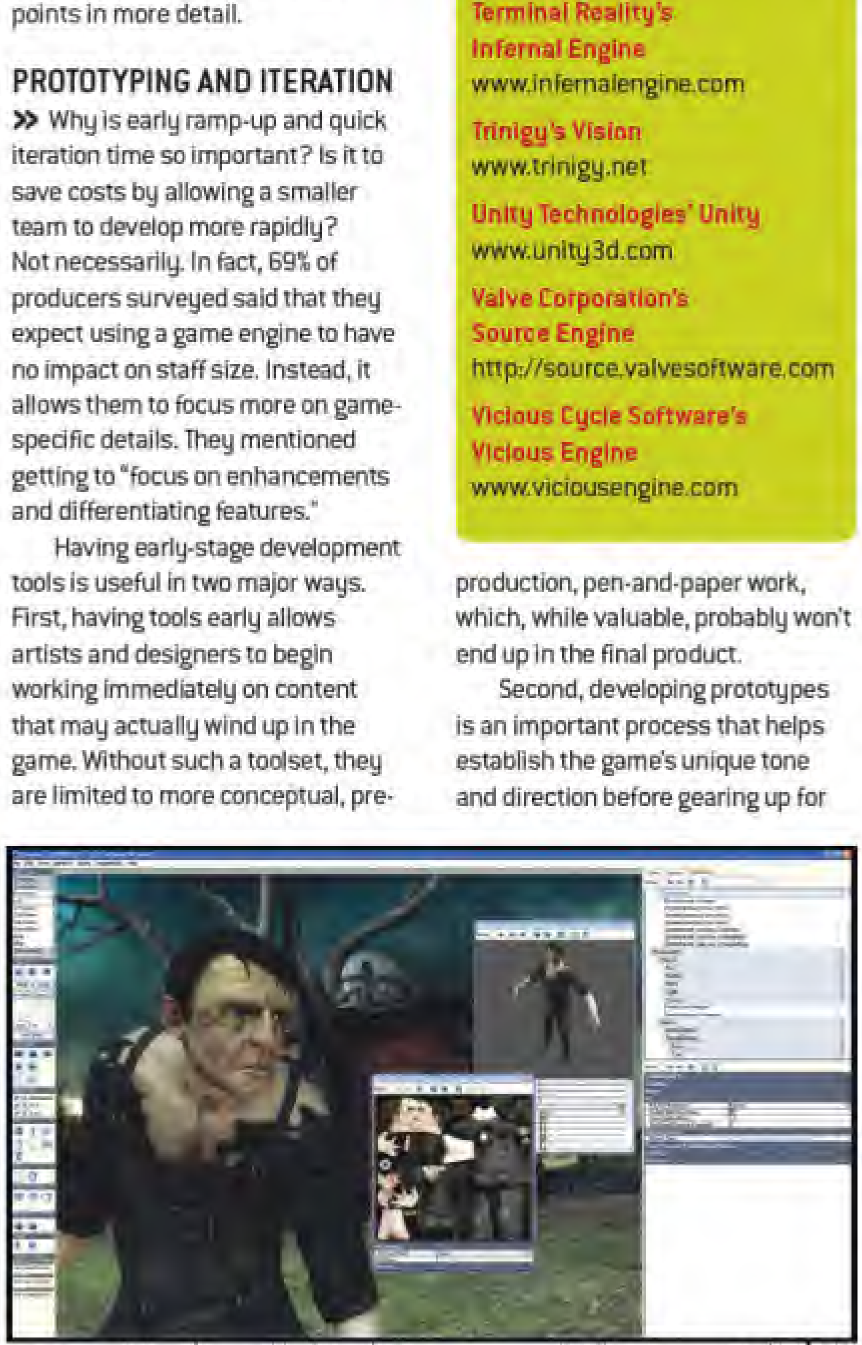
Unity Technologies' Unity
www.unity3d.com

Valve Corporation's Source Engine
http://source.valvesoftware.com

Vicious Cycle Software's Vicious Engine
www.viciousengine.com

table 1 **platform**

FOR WHICH PLATFORMS ARE YOU CURRENTLY DEVELOPING GAMES?	
XBOX 360	70.6%
PLAYSTATION 3	66.2%
PC	55.9%
NINTENDO WII	32.4%
XBOX LIVE ARCADE	14.7%
PLAYSTATION NETWORK	13.2%
NINTENDO DS	7.4%
SONY PSP	5.9%


Evolution engine.

Top: Source engine. Inset left: IdTech. Bottom left: Torque engine. Bottom right: Infernal engine.

experiment? She may become prone to settling for "good enough."

If your game could be previewed on the target platform five minutes sooner, how many more times could your artists check out their work each day?

Drilling down on the idea of rapid iteration, more than 50% of developers surveyed reported that a small change to their game's content or code can be in and running in less than two minutes, with the average iteration time clocking in at around 3.5 minutes. Some console teams use a PC version of their engine or a full-featured world editor with their engine integrated into it in order to achieve speedy iteration times without having console previewing. But rapid preview capability on your target consoles is also an incredibly valuable tool to have in your arsenal, since art can be especially affected by a change in target platform.

For a full content build, the average time taken is much longer—approximately 105 minutes. Typically, a full build is done at night, or when a major change has been made to the code or toolset, and involves code compilation, art and audio format conversion, and occasionally other pre-processing steps, such as lightmap construction and spatial partitioning. While it is valuable to reduce the amount of time required for a full build, if it's done in the evenings, it has less of an impact on day-to-day development. See Table 7 (Pg 12) for the typical length of time needed to do a rebuild.

Of those surveyed, 88% said they use automated build systems or continuous integration to ease their development. CruiseControl.NET was mentioned by many as a popular continuous integration choice.

GAME-SPECIFIC FOCUS

» By purchasing a game engine, developers gain access to an array of technology and a suite of tools. This gives them the ability to immediately begin focusing on their title. The engineering team gets a solid working engine and can begin developing additional tools and technology unique to the game's needs.

I asked executives to rate several tools and technologies, in terms of importance, that they might expect to receive

table 2 awareness

WHICH OF THE FOLLOWING GAME ENGINES ARE YOU AWARE OF?

UNREAL ENGINE	93.4%
CRYENGINE	92.1%
TORQUE	90.8%
GAMEBRYO	85.5%
SOURCE	82.9%
IDTECH	75%
UNITY	55.3%
VICIOUS ENGINE	47.4%
INFERNAL ENGINE	32.9%
VISION	30.3%
EVOLUTION ENGINE	11.8%*

table 3 engine use

WHICH OF THE FOLLOWING GAME ENGINES HAVE YOU USED?

UNREAL ENGINE	60.5%
TORQUE	23.7%
GAMEBRYO	22.4%
SOURCE	19.7%
CRYENGINE	9.2%
UNITY	7.9%
VICIOUS ENGINE	6.6%
VISION	6.6%
IDTECH	2.6%
INFERNAL ENGINE	1.3%
EVOLUTION ENGINE	0%*

table 4 perceived usefulness

ON A SCALE OF 1 TO 5, HOW USEFUL WOULD THE FOLLOWING ENGINES BE FOR YOUR PROJECTS (RESPONSES OF "UNKNOWN" WERE NOT COUNTED)?

UNREAL ENGINE	3.77
CRYENGINE	3.21
SOURCE	3.15
IDTECH	3.12
INFERNAL ENGINE	3.00
GAMEBRYO	2.87
EVOLUTION ENGINE	2.75*
UNITY	2.71
VICIOUS ENGINE	2.45
TORQUE	2.16
VISION	2.00

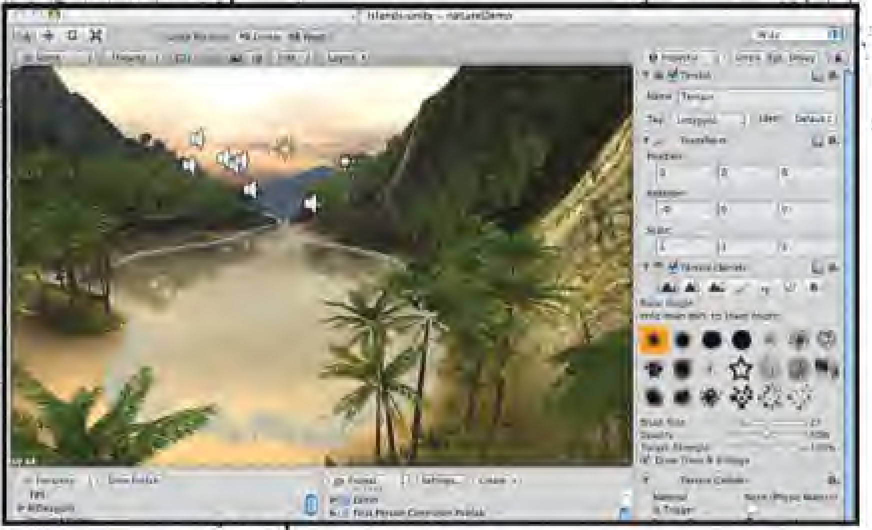
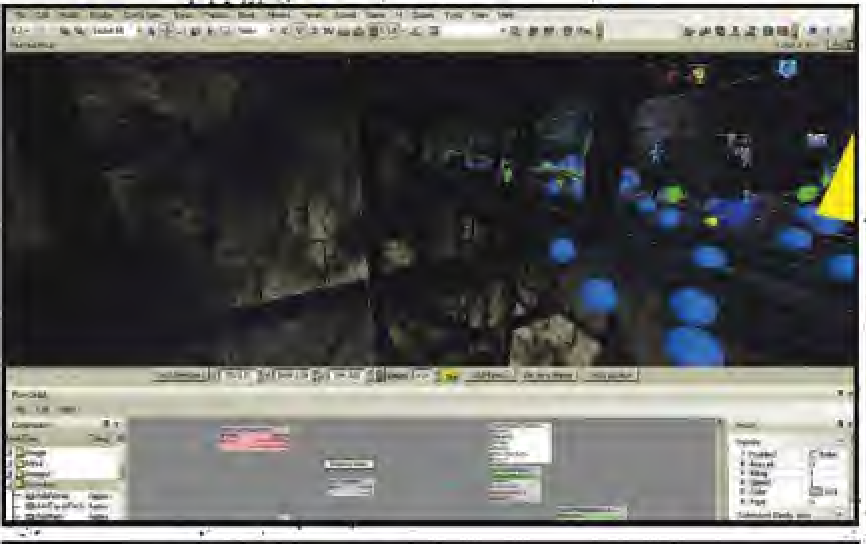
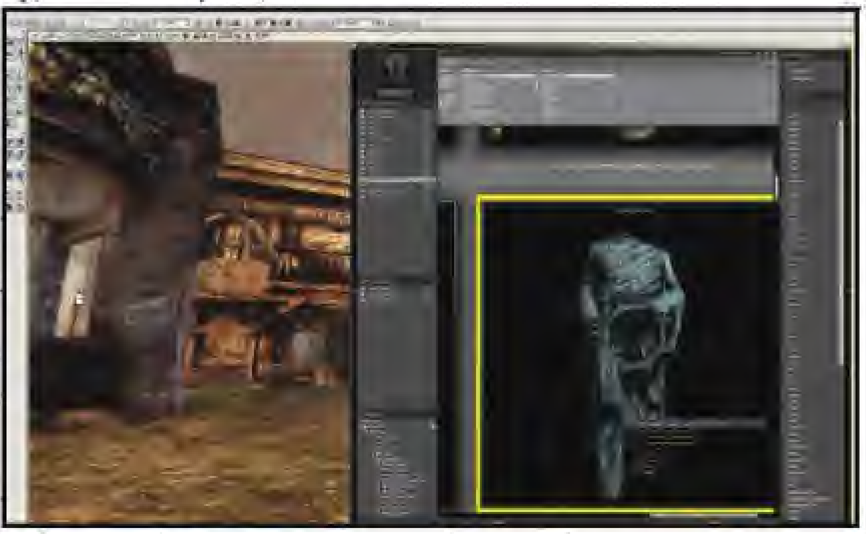
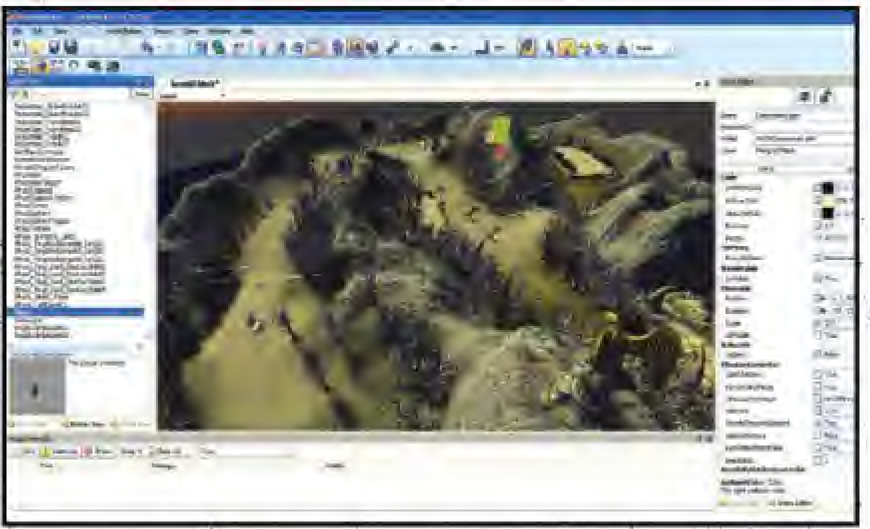
Vision engine

*AT THE TIME OF THIS SURVEY, DIGITAL EXTREMES' ENGINE HAD YET TO BE FORMALLY RELEASED.

Dynamic behavior. Clothing required.
Unparalleled support.
Havok Behavior
Havok Cloth
havok

game engine SHOWDOWN

Top to bottom: Unity, CryENGINE, Unreal Engine, Gamebryo.

when purchasing game engine middleware, as well as what middleware libraries they are most interested in integrating into the engine (see Table 8).

It's debatable whether profiling capability should be considered a system or tool, but clearly it's incredibly important. If I'm making a game on a third-party engine and can't easily tell why I'm getting 10 fps instead of my target of 30, I'm very likely to make some angry phone calls and figure it's the engine's fault instead of my own. Having live preview on the target, so the content creators can see what they're creating properly and iterate on it quickly, is also very important.

A standalone world builder is something most game engines seem to provide now. However, I've seen many younger game studios use their 3D art packages as world-building tools through the use of custom extensions. But as one respondent said, when it comes to licensing a game engine, "a game editor is perceived as a must-have, as building one from scratch is still heavy duty."

Asked in a separate question if people are using a 3D art package or a custom application for world building, 73.2% pointed to using a custom application. Some replies remarked that while using the art package is sometimes the shortest path, it's not necessarily a good way to go. In fact, for PlayStation 3 and Xbox 360, according to one respondent, "the tools don't handle dense scenes well, so the art package doesn't work terribly well as a world-building tool."

Something that came up repeatedly in the survey results is the importance of ease of integration. Whether it's integration with pre-existing technology and tools or with other middleware libraries, a game engine should be designed in a way that eases inter-communication between components.

While a great many developers (46.5%) commented that they would much prefer to create all the tools and technology themselves than license a game engine, many (37.2%) would rather create a low-level engine layer and simply purchase the middleware libraries that suit their game. Only a small number (9.3%) actually want to purchase a full engine suite. This highlights just how important it is for game engine manufacturers to design their engines for integration of game-specific code and other middleware libraries. Engineers almost always prefer creating as much technology as possible for their games themselves.

One executive responded, "It's necessary that there be an existing integration with a middleware library solution. Also, having these is just a start. The flexibility to modify and augment is very important." Another wrote, "I prefer to see an engine have hooks for these systems than to attempt to provide everything."

Of all those surveyed, 90% claim that they are using some type of middleware library, whether to enhance the functionality of a licensed game engine or as part of a completely custom code base. Table 9 shows the libraries that were the most popular.

It would be wonderful if all middleware libraries were easy to integrate into all game engines. But at the moment, apart from a few exceptions, determining whether a particular package can be plugged into your game engine

table 5 **services offered**

ON A SCALE OF 1 TO 5, RATE THE IMPORTANCE OF EACH OF THESE SERVICES OFFERED BY A GAME ENGINE VENDOR.

SOURCE CODE AVAILABLE	4.49
KNOWN TO INTEGRATE EASILY WITH OTHER MIDDLEWARE	4.00
RESOURCE MANAGEMENT EASILY ALTERED	3.85
ONGOING ACCESS TO CURRENT ENGINE BUILDS	3.73
CLEAR ENGINE DEVELOPMENT ROADMAP	3.73

game engine SHOWDOWN

table 6 **prototyping tools**

WHICH TOOLS DO YOU CURRENTLY USE FOR CREATING PROTOTYPES?

C/C++	73.2%
PENCIL AND PAPER	51.2%
FLASH	24.4%
LUA	22%
PRE-EXISTING ENGINE	7.3%
C#/XNA	4.9%

table 7 **rebuild times**

HOW LONG DOES YOUR CURRENT PROJECT TAKE TO DO A FULL REBUILD (ART AND CODE)?

LESS THAN 10 MINUTES	22%
10–20 MINUTES	17%
20–30 MINUTES	14.7%
30–60 MINUTES	17.1%
1–4 HOURS	12.2%
4–8 HOURS	14.6%

table 8 **engine tools**

ON A SCALE OF 1 TO 5, HOW IMPORTANT IS IT THAT THE FOLLOWING TOOLS BE PART OF A GAME ENGINE?

PROFILING SYSTEM	4.19
LIVE PREVIEW ON TARGET FOR ALL TOOLS	4.10
STANDALONE WORLD BUILDER	3.98
PARTICLE SYSTEM EDITOR	3.81
SCRIPTING SYSTEM	3.67
RUN-TIME SCRIPT DEBUGGER	3.58
STANDALONE ANIMATION EDITOR	3.26
SHADER EDITOR	3.23

table 9 **libraries**

WHICH LIBRARIES HAVE YOU LICENSED?

BINK	46.8%
FMOD	43.7%
HAVOK	40.6%
SCALEFORM	28.1%
KYNAPSE	25%

is a slow and complex evaluation process. There are engine manufacturers who provide fairly simple integration steps for some middleware libraries, and they are definitely ahead of the curve. We can only hope that this practice will continue to gain popularity!

Of course, having source code access is fundamental to the evaluation process. The technologists in the survey voted "availability of source code" as the most important practice for game engine manufacturers (followed closely by "known to easily integrate with other middleware"). Since engineers always seek to customize the engine they license for their particular games, having source code access and the ability to recompile the engine are fundamental. Why bother with a binary-only license?

DEVELOPMENT IMPACT

>> As indicated by the survey results, one of the major reasons for buying a game engine is a belief that it will enable game development more quickly since there will be less technology to build. However, what actually occurs on most projects appears to be quite different. In regard to development cycle length, 65.5% of the producers surveyed indicated that using game engine middleware generally results in a project with the same development cycle length—but of higher quality overall: "Building content happens much earlier ... but overall length doesn't change."

Why does the quality improve? According to a majority of respondents, it's primarily due to having rapid prototyping capabilities in the early stages. The top five most likely effects of game engine middleware on the quality of a title as rated by the executives are shown in Table 10. (Very few people responding to the survey expect a lower quality game due to the use of game engine middleware.)

From this data, it seems that a project using game engine middleware is typically scheduled to be the same length as one that is created from the ground up, but the time gained is used to produce a higher quality product. When asked about the potential of a decrease in staff size resulting from the use of game engine middleware, 69% of the executives reported that staff size also remains the same, with a stronger focus on creating differentiating game-specific technology and content.

PRICING

>> At the 2009 Game Developers Conference in San Francisco, game engines were everywhere. New engines were displayed on the expo show floor as well as behind closed doors. The engines roughly broke down into three pricing tiers and corresponding technology levels: 1) high-end with high-technology, 2) mid-priced with many variations in functionality between the engines, and 3) engines with unique business models targeting more casual developers (such as Unity and Torque).

In terms of the price developers expect to pay for a game engine, the survey results varied broadly. On average, the expected cost was 6.1% of the total development budget (with total technology costs at 20% of the total budget). For a multiplatform game with a $15 million development budget, this means the projected cost for a game engine is about $900,000, and $3 million for overall technology development.

Respondents strongly favored (82.1%) flat-rate pricing versus a "lower cost plus royalties on the back end" model. It's certainly nice to have options for financing though, since newer studios are frequently low on cash and may need a royalty-based option. As noted in the survey comments, "We rarely have royalties!" but, "It is best to have both options, depending on projections." One experienced executive noted, "Publishers are averse to getting locked into royalty deals by the developer."

REVVING UP

>> Developers are fortunate to have so many game engines available to license, as there is no one-size-fits-all solution in the game industry! Although there are some similar needs among developers, each game has different needs and different targets, so engines need to be powerful yet flexible. Integrating other middleware libraries into a purchased game engine should be made as simple as possible, and source code availability, support structures, and extensive documentation are absolutely critical.

When game engines first appeared on the scene, many developers complained that they would make all games look alike. In some circumstances, perhaps this was true. But teams have adapted and now recognize that a game's uniqueness isn't based purely on its engine. Licensing an engine is just the beginning of a process. Fine-tuning it with your own code and content, and integrating other middleware libraries as appropriate, can result in a game that is just as distinctive as if you had developed all the tools and technology in-house. And you might even save a little money.

MARK DELOURA *is a video game technology consultant residing in San Francisco, California. He has held leadership roles at Sony, Nintendo, and Ubisoft, and as editor-in-chief of* Game Developer *magazine. Email him at* *mdeloura@gdmag.com.*

table 10 **effects of using a game engine**

WHICH OF THE FOLLOWING RESULTS IS MOST LIKELY WHEN A LICENSED ENGINE IS USED TO DEVELOP A GAME?

HIGHER QUALITY DUE TO RAPID PROTOTYPING	51.7%
HIGHER QUALITY DUE TO RAPID ITERATION	34.5%
HIGHER QUALITY DUE TO GAME-SPECIFIC CODE FOCUS	31.0%
NO IMPACT ON GAME DESIGN QUALITY	24.1%
HIGHER QUALITY DUE TO LIVE PREVIEW	17.2%

We are tired of stupid zombies populating games.
Help us and give 'em brain. Sign up NOW for FREE
and download the NEW xaitment BrainPack SDKs

Set up and control complex game logic in a few steps. Generate your perfect navigation mesh with a single click. Create realistic behavior in no time.

Join the xaitment community now and turn your idea into a stunning prototype without paying any license cost. By signing up for free, you'll receive our complete modular AI Engine plus our world-class support. Experience the future of next gen game technology and work with the smartest AI technology available.

Contact xaitment today for more information about the BrainPack Program under brainpack@xaitment.com or visit our website www.xaitment.com

Give 'em Brain

xaitment
INTELLIGENCE ENGINEERED

Unreal Technology News

by Mark Rein, Epic Games, Inc.

Canadian-born Mark Rein is vice president and co-founder of Epic Games based in Cary, North Carolina.

Epic's Unreal Engine 3 won Game Developer Magazine's Best Engine Front Line Award for three consecutive years, and it was inducted into the Hall of Fame this year.

Epic's internally developed titles include the 2006 Game of the Year "Gears of War" for Xbox 360 and PC; "Unreal Tournament 3" for PC, PlayStation 3 and Xbox 360; and "Gears of War 2" for Xbox 360.

Upcoming Epic Attended Events:

Electronic Entertainment Expo
Los Angeles, CA
June 2-4, 2009

GameHorizon Conference
Newcastle, England
June 23-24, 2009

Develop Conference
Brighton, England
July 14-16, 2009

Please email:
mrein@epicgames.com
for appointments.

POWERED BY UNREAL TECHNOLOGY

BATMAN: ARKHAM ASYLUM PACKS A PUNCH WITH UNREAL ENGINE 3

The door to a part of the DC Universe never seen before outside of comic books is about to be opened: Warner Bros. Interactive Entertainment, Eidos and Rocksteady Studios are using Unreal Engine 3 to bring *Batman: Arkham Asylum* to PC, Xbox 360 and PlayStation 3 on the heels of *The Dark Knight*'s box office tidal wave.

"We have always seen technology as a means to an end, so for this reason we switched to middleware as soon as we moved onto PS3 and Xbox 360 development," said Sefton Hill, director and owner, Rocksteady.

"We evaluated the different options in the marketplace, and Unreal Engine 3 was the best choice for us for two main reasons. First, the creative tools for artists and designers are excellent. Second, the design of the tools is driven by a game development studio which shares a similar philosophy to us – that the creative staff must be empowered to unleash their imaginations to create great results. This gave us every confidence that, as Epic developed Unreal, it would remain consistent with Rocksteady's requirements in the future."

Development started with a team of 40, many of whom had been working with UE3 on various game concepts. That team eventually grew to over 60. When the game ships, total development time will be 21 months from start to finish including pre and post-production.

"Using Unreal allowed us to start work on the gameplay from day one," said Hill. "When creating your own technology, the game team is often waiting for the technology team to deliver their tools before they can start work, which means that even games that have been in development for over two years have often had less than one year of work on the gameplay."

Hill believes UE3's tools remove technology hurdles that restrict many creative professionals from being able to realize their potential. He said the tools are powerful and versatile, making game development about the talent and imagination of artists and designers as opposed to just engine programmers.

"This also frees up your engine team to be much more creative, as well," said Hill. "I know some of our engine team have found ways to use the tools which has surprised even Epic."

Another advantage of using UE3 is its cross-platform interoperability. Hill said each platform has its own inherently unique requirements, so it's important to tailor content to maximize cross-platform results. He added that using UE3 allowed the team to get all three platforms up and running very quickly and easily.

"We have an excellent relationship with Epic, and I cannot speak highly enough of them," said Hill. "They have developed such powerful technology, provided great hands-on support through the Unreal Developer Network, and have been excellent hosts when we have visited them. All this, and they still find time to develop such amazing games!"

Batman: Arkham Asylum

Rocksteady's team makes full use of UE3 technology to bring Batman's spectrum of dark environments to life.

"This universe is so rich and diverse; we set ourselves the goal to build a game world with the Batman DNA flowing through its veins," said Hill. "A key reason to set the game on Arkham Island was to have the player see and feel the history of this universe as they play."

"Our overall goal was simply to deliver an authentic Batman game that was great fun," said Hill.

Batman: Arkham Asylum is the third game featuring the Caped Crusader that's powered by Unreal Engine 3. Midway Games released *Mortal Kombat vs DC Universe* last year, and Sony Online Entertainment is developing the massively multiplayer online game, *DC Universe Online*, for release early next year.

Thanks to Rocksteady Studios for speaking with freelance reporter John Gaudiosi for this story, which will be posted in full at www.unrealtechnology.com.

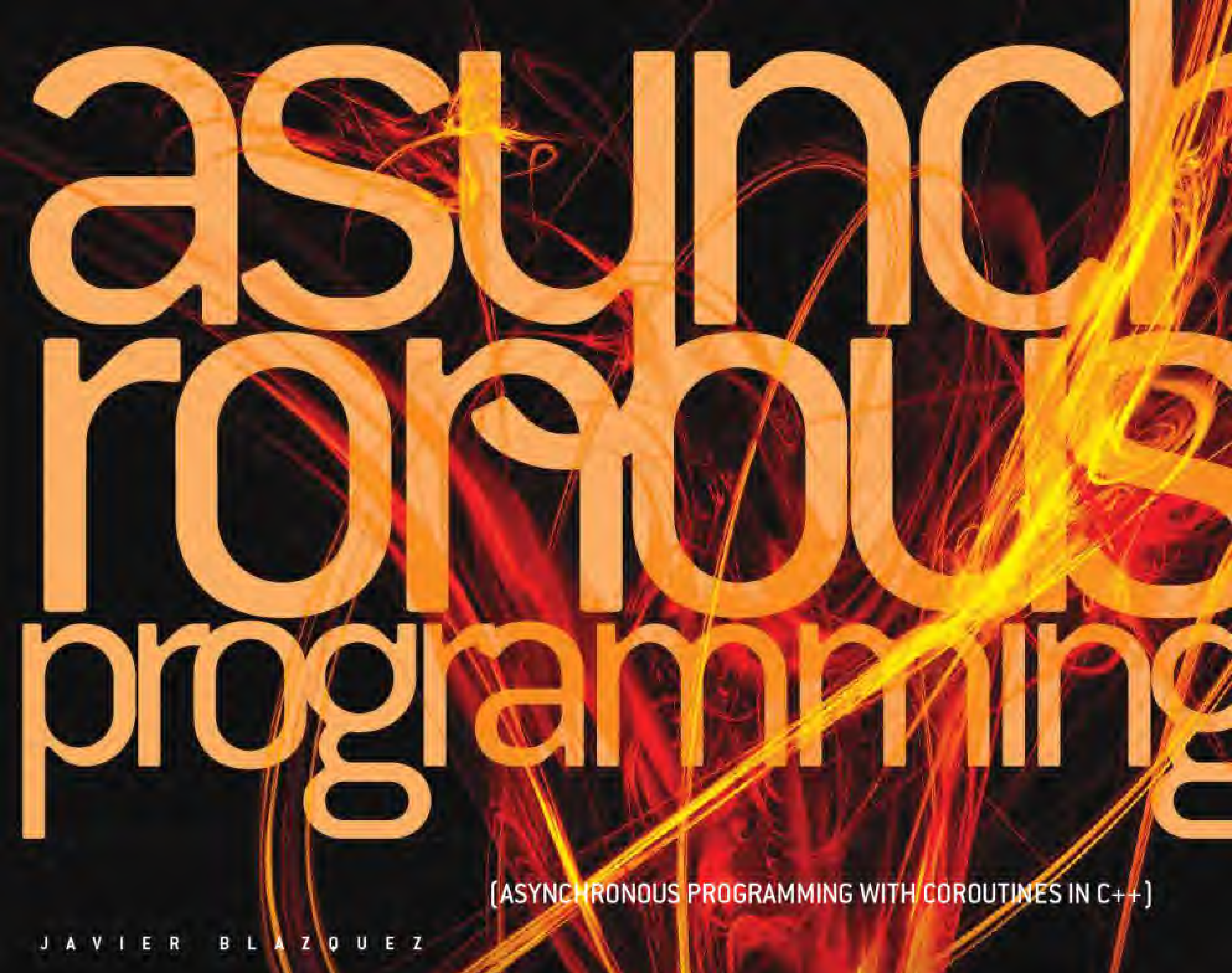
EPIC GAMES

For UE3 licensing inquiries email:
licensing@epicgames.com

For Epic job information visit:
www.epicgames.com/epic_jobs.html

W W W . E P I C G A M E S . C O M

Epic, Epic Games, the Epic Games logo, Gears of War, Gears of War 2, Unreal, Unreal Engine, Unreal Technology, the Powered by Unreal Technology logo, and the Circle-U logo are trademarks or registered trademarks of Epic Games, Inc. in the United States of America and elsewhere. Other brands or product names are the trademarks of their respective owners.



asynchronous programming

(ASYNCHRONOUS PROGRAMMING WITH COROUTINES IN C++)

JAVIER BLAZQUEZ

GOOD RESPONSIVENESS IS A DESIRABLE PROPERTY FOR MANY APPLICATIONS, BUT IT IS especially important in games, where reacting immediately to the player's input is paramount, and dropping frames is unacceptable. Thus, any long process or computation—such as file I/O—should be done asynchronously if at all possible.

However, the traditional model for asynchronous programming requires structuring code in a different way than for regular sequential code, complicating certain kinds of algorithms significantly. Since C++ doesn't provide any built-in constructs for dealing with asynchronous operations, game programmers have to rely on their operating system APIs, which typically provide only low-level mechanisms, such as handles or completion callbacks.

This article presents a method for simplifying asynchronous operations considerably through the use of coroutines.

PROGRAMMING ASYNCHRONOUSLY, THE HARD WAY

>> Non-blocking I/O operations are commonly used in asynchronous programming, so let's use them as an example. When dealing with asynchronous I/O, the typical scenario is to initiate a non-blocking operation and then defer any processing on the data until it becomes available. To know when the data is available, we can wait for a callback or query the handle returned by the asynchronous API.

There are several drawbacks to both options. Relying on a callback forces us to artificially partition the code into multiple functions that perform only part of the computation. When we start an asynchronous request in the middle of an algorithm, we have to pass a pointer to the function that will perform the rest of the computation, and then return from the function immediately, basically delaying the rest of the algorithm while the data is in flight. Once the data is available, the new function will be called to continue the computation. If that function in turn performs a non-blocking operation, it has to similarly defer the rest of the code to another function and pass it as a callback. This approach of chaining functions together can quickly make the code unmanageable and difficult to understand.

Callbacks also have the problem of potentially requiring the code to be thread-safe, since they could fire at any time from another thread. On certain platforms, the operating system invokes these callbacks from the background thread that performs the I/O operations, forcing us to protect any shared data that our callback function might access. On other platforms, like Win32, the completion callbacks are called in the context of the thread that initiated the request, solving the multithreading problem, but they can only run when the thread is in an alertable wait state. If we are not careful to enter this state regularly, we could be delaying the reception of callbacks for a long time.

On the other hand, when polling regularly for completion, we are usually required to make the code maintain some kind of state to know what pieces of data we have processed already and which ones we are still waiting for, effectively turning our routine into a state machine. Again, this approach usually obscures the intent of the algorithm, and has us jumping through hoops to gather the data we need.

To illustrate the point, consider an entity class that represents an object in the world. These entities contain a reference to their 3D model and a list of textures. When loading an entity, we might have some kind of index file that describes it, containing the names of the

model and texture files that it uses. To create an entity, we start by reading the index file, extracting the file names of the other resources, and then loading the data from those files.

To load an entity asynchronously, we would need to write a function similar to that in Listing 1. This function needs to be called regularly (once every frame, for instance) to query the status of the pending asynchronous requests. Note that the function has to not only maintain a notion of current state, but also use member variables to hold the intermediate data used throughout the load, since it will be called over several frames and would otherwise be lost. This example is very simple and could be improved by tracking multiple requests at the same time, but it's meant to illustrate the problem of dealing with asynchronous requests that depend on the results of other requests.

The resulting code is far from ideal, since it turns simple sequential code into a switch statement, which maintains many pieces of state. It also makes error handling especially difficult should a problem arise on an intermediate step.

Given these drawbacks, it's worthwhile to consider alternatives to the traditional asynchronous model. I will show how to use coroutines to avoid dealing with these kinds of callback chains or state machines in the code.

WHAT ARE COROUTINES?

>> Coroutines are basically subroutines that can be suspended during their execution and resumed at the same point later. While subroutines can only give control back to the calling code by returning, coroutines can yield at any point, effectively pausing their computation temporarily. When yielding, coroutines capture the values of local variables, which become available again and can be used normally when execution resumes.

Coroutines are first-class citizens in some languages, but C++ does not support them directly. However, many operating system APIs offer similar constructs, like fibers on Win32 and user contexts on POSIX.

The way these implementations work is by basically performing a context switch, much like what happens when a new thread is scheduled by the operating system. In essence, this context switch just copies out the value of the CPU registers (including the stack pointer) and restores the values for the new coroutine, resuming at the point where it last yielded execution. And because the stack pointers are switched, the resuming coroutine can access its local variables normally. Note that, unlike a real thread switch, this kind of context switch doesn't involve a call into the operating system kernel, making it faster.

For most cases, it's best to use the coroutine APIs provided by the operating system, which are usually quite lightweight. Implementing a homegrown context switching logic can be very tricky on certain platforms. It can be tempting to build a custom system so that we can, for instance, control where the stack for each coroutine is allocated or track its lifetime—however,

listing 1

```
bool Entity::UpdateLoad()
{
    switch (mState)
    {
    case InitialState:
        mFile    = File(mIndexFile);
        mRequest = mFile.Read(&mIndex, sizeof(mIndex));
        mState   = ReadingIndex;
        break;

    case ReadingIndex:
        if (!mRequest.IsComplete()) break;

        // The index has finished loading, start reading model
        mFile    = File(mIndex.mModelFile);
        mModel   = std::malloc(mIndex.mModelSize);
        mRequest = mFile.Read(mModel, mIndex.mModelSize);
        mState   = ReadingModel;
        break;

    case ReadingModel:
        if (!mRequest.IsComplete()) break;

        // The model has finished loading, start reading textures
        mTexIndex = -1;
        mState    = ReadingTextures;
        break;

    case ReadingTextures:
        if (!mRequest.IsComplete()) break;

        if (++mTexIndex == mIndex.mTextureCount) {
            mState = Finished;
            break;
        }

        mFile                 = File(mIndex.mTextureFiles[mTexIndex]);
        mTextures[mTexIndex]  = std::malloc(mIndex.mTextureSizes[mTexIndex]);
        mRequest              = mFile.Read(mTextures[mTexIndex],
                                           mIndex.mTextureSizes[mTexIndex]);
        break;
    }

    return mState == Finished;
}
```

listing 2

```
size_t File::Read(void* buffer, size_t size)
{
    OVERLAPPED overlapped = { 0, 0, { mOffset, 0 }, NULL };
    ReadFile(mHandle, buffer, size, NULL, &overlapped);

    while (!HasOverlappedIoCompleted(&overlapped))
    {
        ResourceManager::Yield();
    }

    DWORD bytesRead;
    GetOverlappedResult(mHandle, &overlapped, &bytesRead, FALSE);

    mOffset += bytesRead;
    return bytesRead;
}
```

one of the properties of a stack is that it grows automatically when the current call chain requires more storage. Because this process is usually controlled by the operating system, it's difficult for the application to hook in its own logic here. We would need to use guard pages or a similar mechanism to capture accesses past the end of the stack that must trigger an extension.

Also, allocating the stack manually can confuse the operating system and subtly break certain constructs. For instance, on Xbox 360, exception handling doesn't work properly for code that runs on a user-allocated stack. If the stack is not allocated by the operating system, then it refuses to run the exception handlers that have been set in the code. Exceptions can be thrown, but they cannot be caught. Even if you don't actually use exceptions in your game, there are certain operating system APIs that rely on them, and they would cease to function properly. **ReleaseSemaphore** is an example of a function that internally relies on exceptions for handling certain cases.

ENTER FIBERS

>> The entity loading code presented earlier can be simplified substantially through the use of coroutines implemented on Win32 using fibers. Fibers are defined as lightweight units of execution that run cooperatively, that is, they are not preempted by the operating system but rather have to yield execution explicitly. Although I show here what an implementation for Windows and Xbox 360 could look like, the concepts are transferable to other platforms.

Consider a **ResourceManager** class that centralizes the creation and loading of resources such as an entity. We would call a **LoadResource<R>** function to kick-start the creation of a resource given its type **R** and the path to its index file. This would create a fiber that, when scheduled, will call a static **Create** function on our resource, which contains the logic for loading and instantiating that particular resource. Note that on a synchronous implementation, we would call the **Create** function immediately as part of the **LoadResource** function, and the function would block until the resource has been loaded, returning a pointer to it.

However, in the asynchronous case, the **LoadResource** function just creates the fiber and returns immediately. Since the resource doesn't exist yet, we cannot return a pointer to it. We have to return some kind of handle or descriptor that will let the client retrieve the actual resource when it's ready. The handle could be as simple as an integer ID that can be resolved later to retrieve the resource pointer. A more interesting handle type would be some kind of strongly-typed proxy object that can be queried to see if the resource is available, as well as provide reference counting semantics and regular pointer syntax for accessing the resource.

Coroutines are first-class citizens in some languages, but C++ does not support them directly. However, many operating system APIs offer similar constructs, like fibers on Win32 and user contexts on POSIX.

listing 3

```
Entity* Entity::Create(const char* filename)
{
    // Load index
    File indexFile(filename);

    EntityIndex index;
    indexFile.Read(&index, sizeof(index));

    // Load model
    File modelFile(index.mModelFile);

    void* modelData = std::malloc(index.mModelSize);
    modelFile.Read(modelData, index.mModelSize);
    Model* model = new Model(modelData);

    // Load textures
    std::vector<Texture*> textures(index.mTextureCount);
    for (size_t i = 0; i < index.mTextureCount; ++i)
    {
        File textureFile(index.mTextureFiles[i]);

        void* textureData = std::malloc(index.mTextureSizes[i]);
        textureFile.Read(textureData, index.mTextureSizes[i]);
        textures[i] = new Texture(textureData);
    }

    return new Entity(model, textures);
}
```

After requesting the creation of one or more resources, we would need to call an **Update** function on the resource manager at some regular interval, for example, every frame. This **Update** function is responsible for processing the pending resource loads, and it does so by basically scheduling the existing fibers in turn, giving them a chance to execute part of the resource creation code. Note that the resource manager cannot preempt the running fiber, so it has to wait until it yields explicitly.

The way each update step works is as follows. First, we call **ConvertThreadToFiber** so that we can start scheduling fibers on the current thread. Next, we switch to the first fiber on the list. The fiber will then resume execution at the point where it last yielded. If it is a newly created fiber, it will start at the main fiber routine, which will immediately call the static **Create** function on the resource to run the loading logic for that resource. Eventually, this function will either return (meaning that the load is complete and the resource is ready) or yield by calling a **Yield** function on the resource manager, allowing the next fiber to be scheduled. Note that fibers don't return from their main routine—they just mark themselves as finished and yield one last time, which effectively prevents the fiber from being scheduled again in the future.

The manager keeps activating the next available fiber until the last one yields, after which the **Update** function cleans up by calling **ConvertFiberToThread** and then returns. The call to **Update** is therefore a single pass over the list of fibers on which they are given an opportunity to progress and maybe finish the load process.

This description might seem to imply that the resource loading code must have explicit knowledge of fibers and yield manually at certain points during its execution. This is not necessarily the case, and I will show how this behavior can be abstracted so it doesn't introduce such requirements for the loading code.

SEQUENTIAL ASYNCHRONY

>> The key enabling feature of fibers here is the opportunity for resource loading code to be written in a sequential manner even though its operations are all asynchronous. Since fibers can yield at any point and be resumed later, they can effectively block on an asynchronous I/O operation until it has completed. Note that this doesn't imply blocking the whole thread of execution—other code can and will run in the meantime on the same thread.

The behavior can be encapsulated in a simple fiber-aware `File` class that presents a synchronous I/O API to the resource loading code but performs asynchronous operations under the hood. For the user, a call to the `Read` function appears to block until the data is available, and indeed it will only return when the data has been read. In the meantime, it will yield to give other fibers a chance to execute, and it will keep doing so until it has determined that the asynchronous request has completed. See Listing 2 for the implementation of `File::Read`.

With this infrastructure in place, we can now write the code for loading entities in a much more straightforward way, as in Listing 3. The new version of the code resembles very much what one would write for loading data synchronously. The code relies on the `File` object to perform the reads, allowing it to be completely oblivious to the fact that it's actually running on a fiber and being scheduled out every time it performs a read.

Additionally, this code can be run unchanged for performing both synchronous and asynchronous resource loads, if we want to give this option to users of the resource manager. All we would need to do is make the `ResourceManager::Yield` function check whether it's being called from outside of a fiber. If it is, it will yield the thread by calling `Sleep(0)`, effectively turning reads into blocking operations.

Another advantage of fibers is that they all execute in the context of a single thread. They do not run concurrently with any other code that executes on that thread. Therefore, the resource loading code can freely access any other data or systems without having to make them thread-safe necessarily. Also, the call to `Update` on the resource manager happens at a predefined place in the game loop, making it easier to reason about the state of other objects at that point. Compare this with the non-deterministic behavior of callbacks that can be invoked at any time on a different thread or whenever the thread happens to enter an alertable wait state.

PAUSE AND CANCELLATION

>> One of the interesting advantages of performing resource loads using fibers is that they can be interrupted at certain points, meaning we don't have to wait for the whole process to be completed in a single call. We can leverage this feature to easily add support for pausing and canceling resource loads, which can be valuable for systems that deal with many resources at the same time and must prioritize or discard some of them for memory, performance, or game-related reasons.

Implementing pause is straightforward. We can extend the resource manager interface with two functions to support pausing and resuming ongoing resource loads. Considering that the resource loading code will yield at some point (when performing a read, for example), we can use that opportunity to check whether the user has requested the fiber to be paused. If this is the case, we can mark the fiber as paused and avoid scheduling it again until somebody requests it to be resumed. Hence, we can effectively delay a certain operation indefinitely.

On the other hand, to properly support cancellation we have to extend the resource loading code so that it regularly checks for this condition. We could provide a function that the loading code can call to check whether it has been canceled and is supposed to return. This function could also act as a checkpoint by yielding execution so that the fiber doesn't run for too long when performing a complex operation. By inserting calls to this function throughout the code, the resource loading function can further improve the responsiveness of the game by reducing the maximum amount of time that the main thread is blocked. These calls can be thought of as preemption points for the fiber.

To implement cancellation support, the code must check the return value of the function and act accordingly. If it determines that it has been canceled, it must clean up all intermediate resources and return a null pointer from the `Create` function. Note that just removing the fiber from the list of running fibers when it has been canceled is not a proper solution because it prevents the resource loading code from cleaning up any intermediate objects, resulting in memory leaks.

An alternative solution would be to throw an exception when a fiber yields and we detect that it has been marked as canceled, and then catching that exception on the main fiber routine. This solution would force the resource loading function to return, cleaning up all intermediate resources as long as the user has properly followed the Resource Acquisition Is Initialization (RAII) pattern, which ensures that resources are cleaned up when they go out of scope by releasing them in the destructor of stack objects. The advantage of this alternative is that it doesn't require the user to add explicit checks for cancellation throughout the code, but relies instead on the proper application of the RAII pattern and having compiled support for exceptions, which is commonly disabled for games.

THE COMMENDABLE COROUTINE

>> I have shown how to leverage coroutines with little effort to simplify code that deals with asynchronous processes, such as file I/O. The resulting code is much easier to understand since it retains its sequential nature and avoids multi-threading issues that can happen when we rely on callbacks.

There are several drawbacks to this approach, of course. The main problem is that coroutines are collaborative by nature. They have to voluntarily yield execution for other coroutines to have a chance to run. Since there is no preemption, a coroutine that's behaving badly can hold the thread of execution indefinitely, a problem that can't happen with the common multi-threaded approach, since the operating system scheduler makes sure that all threads have a chance to run (unless they have different priorities).

There's also a risk of introducing too much latency on I/O operations or reducing their throughput considerably, seeing as coroutines are scheduled out as soon as they initiate an I/O operation, which could complete long before the coroutine is scheduled again. This risk can be partially mitigated by scheduling the coroutines more frequently and performing certain I/O operations synchronously when they are deemed small enough not to block for a significant amount of time.

JAVIER BLAZQUEZ *works as a core engineer at Lucas Arts, where he deals mainly with systems-level libraries for the in-house game engine. Previously he worked at Pyro Studios, where he was responsible for the overall game architecture implementation. Email him at jblazquez@gdmag.com.*

resources

Stackless Python
www.stackless.com

Hawes, David, "Snakes on a Seamless Living World," *Game Developer*, February 2009

Richter, Jeffrey, "Concurrent Affairs" series, http://msdn.microsoft.com/en-us/magazine/cc501041.aspx

ACTIVISION®

GREAT GAMES START WITH GREAT PEOPLE

ARE YOU READY TO EXPLORE THE MOST SOUGHT AFTER JOBS IN ENTERTAINMENT?

Visit our site: activision.com

GAMING THE SYSTEM... HOW TO REALLY GET AHEAD IN THE GAME INDUSTRY

SUCCESS IN GAME DEVELOPMENT TAKES A CAREFUL BALANCE OF HARD work, timing, and company politics, whether we like it or not. And yet, getting ahead in your specific discipline doesn't always mean doing what you're told or following advice to the letter.

A few successful developers with some strong, honest opinions have agreed to share their thoughts on what it actually takes to get ahead in the industry. Every level of employee is represented, from the most junior to the executive level. Pseudonyms have been used to allow the authors to speak their minds unconditionally.

Each company is different, of course, so some of the advice that follows may feel like it doesn't pertain to you—or maybe you disagree entirely. But chances are, what these developers have to say will resonate, even when it comes from a discipline other than your own. To that end, different positions are represented as well, from the "learn the rules of the game" approach of the production entry, to the "do it your own way" approach by the design author.

Don't like what you read? Send your alternate approaches to bsheffield@gdmag.com. The most convincing (or contentious) responses may be published on Gamasutra.com. —*Brandon Sheffield*

PROGRAMMING

JUST DO IT! CONSTRUCTIVE DISOBEDIENCE

By Larry Hacker

There are many things you can do to advance your career as a game programmer. You can excel at the tasks you are given, you can learn new skills, you can research current techniques, you can document your code, you can stay late nailing down a tricky bug, you can follow coding conventions, and you can help others solve problems.

But all these things are simply doing what is expected of you. Let's step outside the box. How can you advance your career by doing things that are not expected of you, or even things that you've specifically been told not to do?

As a programmer, it's not uncommon to see problems that you think should be fixed, or to see an opportunity to improve some piece of code, or speed up a process that takes a lot of time. It's also not uncommon for your suggestion to be ignored, or dismissed with an "it's not broke, so let's not fix it" response.

Say your code uses a lot of hard-wired checksums as identifiers. Every time a new identifier is added, the programmers use a command line utility to calculate the checksum, and then copy and paste it into the code. Now, it would be vastly quicker if they could do this inside the editor with some hot key. You suggest this to the lead, and he says, "We don't have time for things like that."

What should you do? You should just do it—on your own time. Figure out the macro system in the editor, hook in the checksum generator, and link it to a hotkey. Then quietly show everyone what you've done. The other programmers will be grateful that you've saved them work and will be impressed with your coding. And the lead will hopefully admire your initiative.

I say "hopefully" because the "just do it" approach is a potential minefield. While it's a great opportunity, you do need to be careful that you know what you're doing. Before taking the initiative (or rather, before telling people that you did), make sure it's really something worth doing. If possible, try it covertly first so that if it's not actually worth doing, nobody will need to know you wasted your time—and make sure that "your time" is actually that. People have different opinions of what your own time is and might think any time you spend coding should have been company time.

"Oh, that? It only took 10 minutes!" That line usually absolves you of the time-wasting label, and makes you look even more impressive.

ART

A POLITICAL PARTY

By Mr. Confidence

For all you game artists looking to level up in your careers, here is a compendium of suggestions that might help you do just that.

Ask for forgiveness. There comes a time on every project when you know the solution to a problem, but you haven't asked permission from your lead to implement it. Sometimes, it's better to just do it and apologize than to request permission. You could do worse than getting a warning about going over your boss' head, but solving the problem will likely diffuse her anger.

Be the single point of failure. This sounds contrary to what you'd want to be, but it suggests increased accountability and ownership. This is exactly what you want! Make the team dependent on your skills. If you "own" part of the production process, then they can't live without you.

Introduce first. Well before the end of your project, you should spend time investigating other projects at your studio. Take time to meet people on the project that you want to be on and carve your own destiny. Don't wait for your lead or HR representative to tell you where you're going next. If there's one thing makes an impact on a team, it's passion for their project.

Be the hero. During production or even prototyping, always be on the lookout for opportunities to make a big impact. Finish early. Create a process. Solve an art issue. Whatever it is, nothing beats the kudos that come from being the person who saved the day.

Meet the world. Too many talented artists are content to sit at their desks and work hard as great employees, but you are doing a disservice to yourself and to your studio if you aren't making an impact outside the office as well. Opportunities to speak at the Game Developers Conference and other venues offer an opportunity to network and communicate with the industry as a whole, and the potential opportunities this leads to can be plentiful. One good art talk generally leads to more.

Circumvent the boss. This is the sneaky, dark secret. Honestly, if your boss is not helping you advance your career, or if he's making bad decisions that you think will adversely affect the project, communicate that knowledge upward. Most of upper management is insulated from knowing what's happening on the project on the team level. You may do more than just help the project—you could also be helping your career in the process. Just make sure you express yourself in a professional manner, and don't sound like a complainer.

DESIGN

SHOW AND TELL

By Daxter Crate

As a game designer, your job is to design games. As obvious as that sounds, it's easy to lose sight of and get sucked into having the wrong priorities.

You have to contend with company politics, unreasonable requests from publishers, the stroking of egos, and other baloney that has nothing to do with the game itself.

I recommend taking a vow to make the game the best game it can be, no matter what that means for all that other hogwash. Little Jimmy in Iowa who buys your game doesn't know or care about any of that other stuff, and neither do game reviewers. They judge the game you put in front of them, so put the best game in front of them you can.

On one project I worked on, an outside art contractor we were using created an elaborate standoff about fixing art bugs. My company wanted the contractor to fix all the art bugs on principle. Nice principle, but Little Jimmy and all the other Iowans only care if they are fixed, not which politics prevented them from being fixed. I personally fixed several and recruited an artist co-worker to fix more on his own time.

In a different situation, we wanted to add a set of sound effects but had no one allocated to do the sound processing. I downloaded free sound processing software and learned how to do it myself because I knew it would improve the product. These anecdotes aren't even about game design, but they help create a culture where "make the product good" is the highest priority. If you can get other team members to buy into this mindset, your team as a whole will be capable of making that much better of a game—and that is how you'll be measured in this industry.

In addition to doing good work, try to let the general public know exactly what you're doing. As a designer, your decisions shape what the player-experience is. Players will be very interested to hear why you made those decisions, and that raises your value in the industry outside the company.

The reality is you're probably not going to be at the same company your whole career. (Although if you do work at an awesome company, staying put could be great!) It's to your advantage to let the outside world know exactly what you did. Your company or publisher might not want to see you self-promote because they might see an advantage in preventing you from getting credit for your work. Fair is fair though, and what you have on your side is that the marketing and even the design of your game will benefit from keeping players in the loop. Make that argument if you get any resistance from within, and try to let the world know what it is you actually do.

PRODUCTION

THE UNSPOKEN RULES

By Tracky McProject

It might sound crazy, but sometimes, shipping something awesome on time and on budget isn't enough.

Don't forget that as a game producer, you are judged on not only the results you get, but also how you went about getting them. There are a lot of ways to interpret the job of a producer, and one thing you'll want to do early on is make sure you're doing things the way your boss imagines them being done. A pitfall for people in the production field is finding out after the fact that their production style or methods weren't what the powers that be actually wanted.

And don't expect them to tell you what they want right off the bat, either! I've seen them wait until the project is done and it's employee review time for it to finally come to light. So before you get in too deep, spend some time learning what the boss wants in terms of process. If she doesn't seem to care either way, don't believe her. It will come back to haunt you later.

Another thing to watch out for is unspoken rules. Unfortunately, nobody will tell you about these upfront. I once watched my boss in the production organization give some creative feedback on the game's story, and I naively assumed this was an acceptable practice. My own small attempt at creative feedback turned out to be a strike against me when it came time for a performance review! My boss explained that he had a rapport with the creative director that I didn't have. Okay then, lesson learned!

Most of the time, you find out that these rules exist only by breaking them, but sometimes you can spot them from someone else's turmoil, such as when the co-founders of a company are fighting. Be very careful when sending emails on sensitive topics like these. Everything, including your language and who is on the To and CC line, can turn into a landmine. When in doubt, talk to folks in person. If you break one of these unknown rules in conversation, well, at least there isn't a permanent record of it.

Finally, don't forget why you're putting up with all this crap to be a producer. For me, it's the guys on the team. One time, the company I worked for bought a limited number of new monitors and decided to dole them out based on tenure. It happened to be my turn to receive one. As the IT guys installed this big, brand new screen at my workstation, I couldn't help

but think about an animator we had just hired, fresh out of school. He was working hard and doing amazing things in Maya using small, crappy monitors. How could I look up from an Excel spreadsheet on my beautiful wide-screen display while this poor kid who actually made content was struggling with 15 meager inches of visual real estate? "Hey, give my new monitor to him," I told the IT guys, who were happy to comply. Pull for your guys like this, and pretty soon you'll have their trust. That might not get you a promotion immediately, but remember, these are the people who actually make the game.

AUDIO

SHUT UP AND LISTEN!

By Johnny Foley

Audio always gets forgotten. It always ends up being last—in everything! Meetings, production, you name it!

It makes sense, then, that audio developers, be they audio directors, composers, sound designers, or voice directors (all sensitive souls who strive for quality), are generally evangelists by nature. At any audio-exclusive gathering, you'll hear the same war stories and tales of producers and senior management making horrific decisions that adversely affect the quality of the audio.

Venting is natural and, I would argue, actually necessary to mental well-being and survival in video game audio development. However, that same airing of grievances can often be the downfall of audio developers who cross over to the dark side and start overtly speaking their minds and generally losing it with senior development staff.

I have seen and interviewed many of the fallen, those who were let go for disagreeing with a fundamental product or studio decision because it affected their audio in some disagreeable way, and who, rather than working in a bustling team atmosphere, now work in a lonely home studio.

Developing audio for a video game is, ironically, intensely collaborative, not just between the audio, design, art, and code, but (and this is something that never gets talked about) between producers, senior producers, and executive producers. The top brass often likes to be involved when it comes to casting, dialogue recording, and directing, and sometimes composing There's always a corporate creative guy who owns a studio and thinks audio is his "thing." There will always be an exec producer whose "thing" is dialogue direction, especially if someone famous is involved.

The same thing happens when product and marketing people get involved, too. And, oh yes, they are also your collaborators. Marketing and PR always had a presence in voice casting meetings, and often it was my job to fight for quality and common sense casting, while they bounced around the latest pop stars as would-be lead characters in the game. The trick is to view this as a part of the collaborative process as an inevitable part of development. Get them involved and listen to their ideas. In all likelihood, they'll be distracted by a shiny object and will leave you alone.

Survival Tip: Always treat senior publisher staff and producers with respect (through all communication channels), as they are collaborators, too. Often, they see the bigger picture on a product with a clarity that you don't have. Listen to their ideas, try to understand what they want to achieve, and give them a way that it can be done. Compromise of creative ideals is unavoidable, but it need not always be negative.

If you can make things happen for them and make what they want actually happen, they will sing your praises and adorn you with all the respect you can hope to get. This will ultimately make future projects a lot easier.

QUALITY_ASSURANCE

GETTING THE HELL OUT OF QA

By Bugsy Checker

QA is often thought of as the standard point of entry into game development for careers outside the programming and art fields. While this is true for some, it also means you're far from alone in trying to make your move. Because of this, the most basic rule for getting the hell out of QA is to get noticed.

Know your producer, and make yourself an asset to her. Find out what extra work needs to be done, and do it. If the company doesn't have an associate producer role, try to forge one by taking on some of those tasks. Get yourself known as the guy who is interested in learning new skills and going the extra mile.

Unfortunately, just being good at your job and eager to learn often isn't enough. You'll need to play politics.

Some companies develop an adversarial culture between QA and the development teams. Do your best to avoid this. It's going to be difficult to join development if you see each other as the enemy. Beyond that, you'll need to know people socially. Be friendly around the office, go to company events, and get to know people in the positions that will be making hiring decisions when the time comes. The smaller your company, the easier it is.

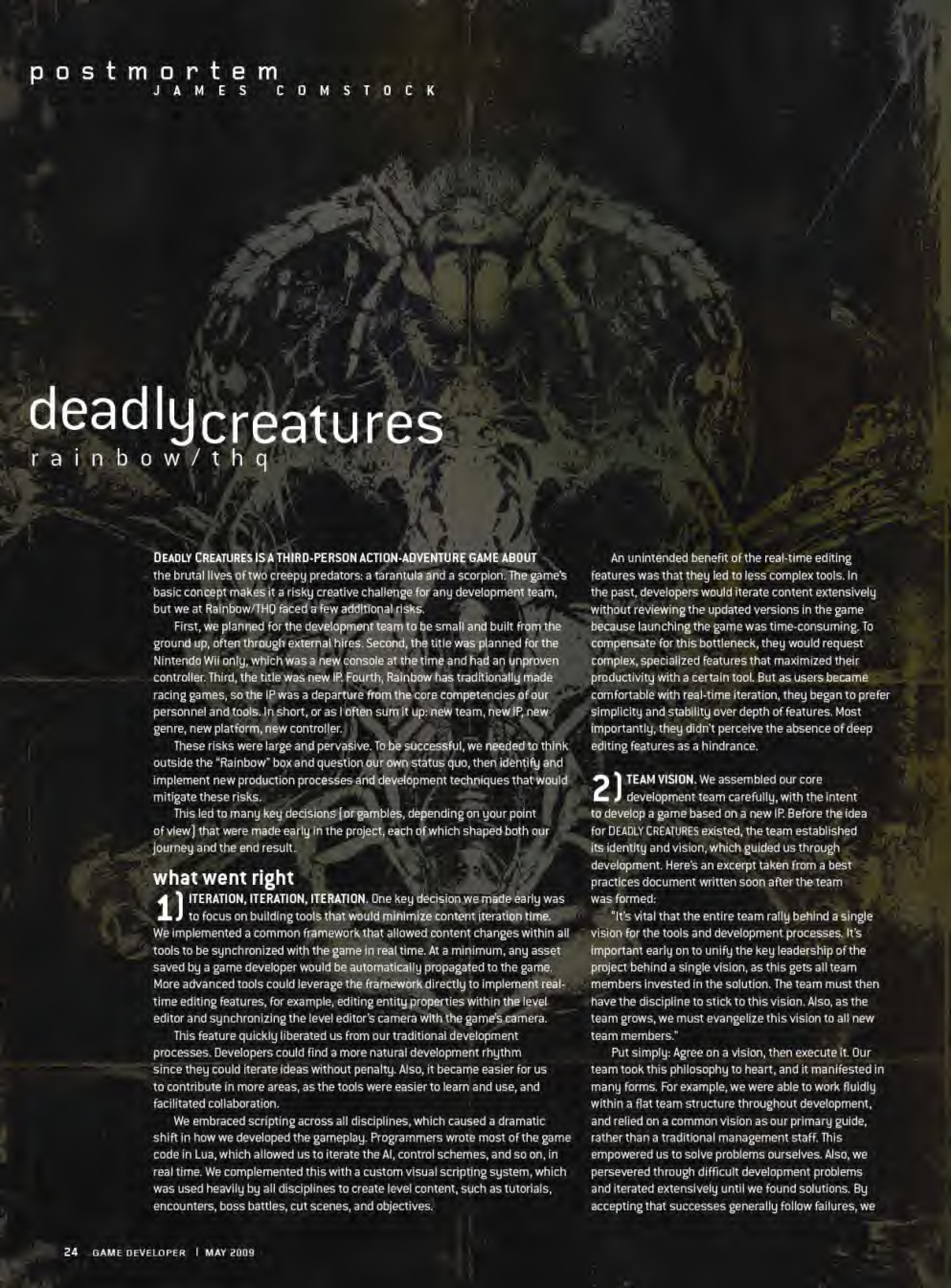
There are also a few things to watch out for. When taking on extra tasks and learning new things, don't do it to the point that you're ignoring your duties in QA. It's also important to not become Free Work Guy. You don't want to be seen as the person who doesn't need to be promoted because, after all, he'll do all the extra work for free. Do as much as you can, but don't hesitate to make it clear that with your QA responsibilities, you can only do so much. And don't overdo the socializing. You want to be friendly and easy to talk to, but not a social butterfly who can't walk to the bathroom and back without chatting for 30 minutes about the last episode of *Battlestar Galactica*.

Assuming you can walk these lines effectively, you should be in good shape, but nothing is sure in this world. While working at a small company makes it easier for a QA staffer to get noticed, the budget there may be too lean to accommodate an associate producer or junior designer position, meaning getting out of QA could require you to jump two or three steps up into a role that's a little out of your league. Larger companies, on the other hand, are more likely to have one-step-up openings, but that also means more competition and more distance between you and the people doing the hiring.

Naysaying aside, QA is still probably the best place to get a foot in the door. You'll develop familiarity with development cycles; working as a lead will give you important experience managing people and schedules; and creating test plans and scripts develops your technical writing abilities—all of which are essential skills for both game producers and designers. And, of course, the longer you're in the industry, the more people you'll know, and the more connections you'll make. Don't assume that the promotion you're hoping for will be at the company where you're currently working.

THE MAD DOCTOR WANTS YOU TO APPLY...
...OR THE GIRL GETS IT!!!
666
TALENTED GAMES FOLK APPLY!
WWW.NEVERSOFT.COM

JAMES COMSTOCK



deadly creatures

rainbow/thq

DEADLY CREATURES IS A THIRD-PERSON ACTION-ADVENTURE GAME ABOUT the brutal lives of two creepy predators: a tarantula and a scorpion. The game's basic concept makes it a risky creative challenge for any development team, but we at Rainbow/THQ faced a few additional risks.

First, we planned for the development team to be small and built from the ground up, often through external hires. Second, the title was planned for the Nintendo Wii only, which was a new console at the time and had an unproven controller. Third, the title was new IP. Fourth, Rainbow has traditionally made racing games, so the IP was a departure from the core competencies of our personnel and tools. In short, or as I often sum it up: new team, new IP, new genre, new platform, new controller.

These risks were large and pervasive. To be successful, we needed to think outside the "Rainbow" box and question our own status quo, then identify and implement new production processes and development techniques that would mitigate these risks.

This led to many key decisions (or gambles, depending on your point of view) that were made early in the project, each of which shaped both our journey and the end result.

what went right

1) ITERATION, ITERATION, ITERATION. One key decision we made early was to focus on building tools that would minimize content iteration time. We implemented a common framework that allowed content changes within all tools to be synchronized with the game in real time. At a minimum, any asset saved by a game developer would be automatically propagated to the game. More advanced tools could leverage the framework directly to implement real-time editing features, for example, editing entity properties within the level editor and synchronizing the level editor's camera with the game's camera.

This feature quickly liberated us from our traditional development processes. Developers could find a more natural development rhythm since they could iterate ideas without penalty. Also, it became easier for us to contribute in more areas, as the tools were easier to learn and use, and facilitated collaboration.

We embraced scripting across all disciplines, which caused a dramatic shift in how we developed the gameplay. Programmers wrote most of the game code in Lua, which allowed us to iterate the AI, control schemes, and so on, in real time. We complemented this with a custom visual scripting system, which was used heavily by all disciplines to create level content, such as tutorials, encounters, boss battles, cut scenes, and objectives.

An unintended benefit of the real-time editing features was that they led to less complex tools. In the past, developers would iterate content extensively without reviewing the updated versions in the game because launching the game was time-consuming. To compensate for this bottleneck, they would request complex, specialized features that maximized their productivity with a certain tool. But as users became comfortable with real-time iteration, they began to prefer simplicity and stability over depth of features. Most importantly, they didn't perceive the absence of deep editing features as a hindrance.

2) TEAM VISION. We assembled our core development team carefully, with the intent to develop a game based on a new IP. Before the idea for DEADLY CREATURES existed, the team established its identity and vision, which guided us through development. Here's an excerpt taken from a best practices document written soon after the team was formed:

"It's vital that the entire team rally behind a single vision for the tools and development processes. It's important early on to unify the key leadership of the project behind a single vision, as this gets all team members invested in the solution. The team must then have the discipline to stick to this vision. Also, as the team grows, we must evangelize this vision to all new team members."

Put simply: Agree on a vision, then execute it. Our team took this philosophy to heart, and it manifested in many forms. For example, we were able to work fluidly within a flat team structure throughout development, and relied on a common vision as our primary guide, rather than a traditional management staff. This empowered us to solve problems ourselves. Also, we persevered through difficult development problems and iterated extensively until we found solutions. By accepting that successes generally follow failures, we

were able to stay focused on moving forward in the midst of trying circumstances.

Our success was contingent on setting high standards for both ourselves and our potential hires. We carefully selected team members who could understand, champion, and execute our team vision. Of course, this slowed the hiring process, but we accepted that our short-term hiring pains would lead to long-term development gains.

3) GENERALISTS PHILOSOPHY. When searching for a new IP, we looked for one that would allow us to keep our team small. We wanted to maintain our team chemistry and to work in a fluid development environment, while also allowing each of us to have a broad impact on the game.

To succeed, each person would have to shoulder more responsibility and take on ancillary roles. We also had to ensure that each of our new hires was willing and able to work in this kind of environment.

Strike teams were used throughout development, which forced each of us to wear many hats. The strike teams were always inter-disciplinary and were generally responsible for developing gameplay and combat mechanics, scripted encounters, cut scenes, and boss battles. Depending on how the strike teams were constructed, individuals would pick up and run with different responsibilities. We even had some talented members of our QA department participate. They went on to create some of the best encounters in the game.

On the programming side, we took this concept a bit further. We abolished the roles of game programmer, technical programmer, and tools programmers, and redefined ourselves collectively as generalist programmers. In our definition, a generalist programmer is responsible for designing and implementing end-to-end features, from the tools through the gameplay. We relied on teamwork to fill in the gaps, as each programmer had areas of expertise that could be tapped when others were faced with challenges.

Hiring programmers became difficult, as many applicants didn't fit the role, or preferred roles that offered specialization. However, the results of this structure were fantastic, as it fostered levels of communication, teamwork, and productivity that exceeded our expectations.

4) CONCEPT ART AS COMMUNICATION. In preproduction, we homed in on a painterly art style that complemented our game design and the technical constraints of the Wii.

Concept art was the linchpin in communicating the art style, setting standards, and measuring quality throughout development. Our goal was to achieve the look and tone of the concept pieces directly within the game experience.

We used a mix of internal and contract artists to visualize a diverse cross-section of the game's environments. Commissioning work from a wide variety of sources helped us to digest and interpret our vision, and allowed us to generate a large number of concept pieces quickly. The final concept pieces fed into all aspects of development and helped us set the tone for the game. They inspired ideas for level design, creature design, and story presentation. We used the pieces to communicate our vision to marketing, sales, product development, and the press.

It's a testament to the talent of our artists that there are many areas of the game that met or exceeded the original concept pieces they were based on, despite the technical limitations of the Wii.

Concept art wasn't just for support and visualization. We continued to use it as a resource throughout production. Our internal concept artists created large and exquisitely detailed texture scripts for every imaginable material: rock, wood, sand, rusted metal. These provided a consistent

yet stylized palette from which our artists could pull to add texture to our diverse environments.

5) REMOVING BOTTLENECKS. Content creation bottlenecks can be costly when they prevent game developers from working. In addition to content iteration time, we identified and resolved two key bottlenecks that were the source of significant downtime in previous projects.

First, a developer may be unable to work because the tools are broken (in this context, we consider the game to be part of the tools). This situation can be especially costly, as one bug in the tools can bring the entire team to a standstill. We believed this to be avoidable and set a goal never to break the tools. We created a formalized tools release process, which the programmers followed throughout the project. In a nutshell, all the tools were released simultaneously on a regular schedule, once a week. The tools were well-tested and documented prior to each release. If a critical bug slipped through testing, we immediately released an update containing a bug fix.

The benefits were many. For one, we consistently released stable tools, which helped to foster a trust between the programmers and other developers. Additionally, the total number of support requests for programmers declined, as the tool user's first instinct was not to blame the tools indiscriminately for every issue that occurred. When an issue did occur, it was easy to determine if the tools contained a bug and which release introduced it, or whether the issue was caused by user error or bad data.

Additionally, a developer may be blocked while waiting to edit a level that is locked by another user. To streamline access to level data, we added support for fragments to the level editor.

A fragment is a file that contains level data. Each level includes any number of fragment files, and each fragment file can be checked out and edited independent of other fragment files. By storing data in multiple fragments in a manner that reflected our workflow, we were able to greatly reduce contention over access to level data.

As a side note, we were also able to include a fragment within multiple levels, which facilitated content authored in one fragment to be used in

any or every level in the game. This was invaluable for authoring shared data for the UI, PC, NPCs, and so forth.

what went wrong

1] CREATIVE CONSTRAINTS. When pitching DEADLY CREATURES, the IP stood out because of how different it was from other products we expected to see on the Nintendo Wii. Many action-adventure games take place in fantasy or sci-fi settings, and those that take place in realistic settings usually have a human protagonist. In DEADLY CREATURES, we aimed to craft a realistic world and believable behaviors for the tarantula and scorpion. This prevented us from leveraging many of the design patterns in the standard action-adventure toolbox.

By choosing non-human, non-speaking, non-bipedal protagonists, we created an emotional barrier that could hinder the player from connecting with a traditional narrative experience. Our solution was to present an overarching story via non-traditional means. As a scorpion or tarantula, the player becomes a mostly passive participant in a story told through the dialog of human NPCs and the details found in one's surroundings.

Adhering to a naturalistic tone significantly affected the gameplay mechanics. The player's expectation is that a tarantula can't beat up insects with a crowbar, and a scorpion can't solve puzzles by pushing buttons. We had to redefine our expectations for melee and ranged combat, as insect combat generally revolves around slow and deliberate grappling rather than the punching, kicking, and throwing. Dealing with puzzles and collections was a challenge, as we strived to avoid any behavior that seemed too intelligent or unrealistic for these creatures.

Level design was tricky, too, since we were designing for creatures that can defy gravity and walk on any surface. A completely open world without any restrictions, progression, or guidance would have been unfocused, so we had to define constraints which directed the player but which still meshed with our design goals. We started to rely on huge setpieces, like a rusted-out truck, which became unique and memorable, but required a lot of creativity to transform into compelling game experiences.

Although we feel that we lived up to the challenges and constraints posed by our core concept, we had to iterate the design throughout development and rework content multiple times along the way.

2] SOLIDIFYING GAMEPLAY EARLY. We recognized the creative constraints we faced, but underestimated the complexity of crafting a balanced gameplay experience. This led us to deprioritize work on the gameplay during preproduction. It's not that the work wasn't a high priority, but we had many other pressing risks to address, such as minimizing iteration time within the tools, developing gameplay design tools, updating our tools to support the Wii, and researching the Wii remote. We distributed resources to tackle this broad set of risks early. In retrospect, this left our gameplay prototyping efforts understaffed for much of preproduction.

This problem had a trickle-down effect on the rest of development. We began production before all our gameplay was solidified, which forced us to approach some development tasks out of order. For example, we began building all the levels before the gameplay mechanics were finalized, which led to bottlenecks and rework as mechanics changed. Consequently, the project plan fluctuated rather significantly, as unplanned rework required us to reorganize tasks and resources to minimize the negative impacts. We also weren't able to balance the user experience until later in the production, which forced us to refactor or scrap content that had already seen significant development time.

Allocating resources differently during preproduction may have led to better results—or even worse. It may be the case that we simply tackled too many risks with the project, given the resources and time available. However, there are many smaller decisions we could have made that, cumulatively, would have improved the efficiency of how we developed the gameplay.

3] BUILDS. During preproduction, we decided to delay putting the game on DVD in order to free up resources for other tasks. At the time, we didn't have enough resources to address all our key risks, and this was a calculated risk that seemed reasonable given the alternatives. We also planned to deliver most milestone builds in a rough, in-development format rather than in a polished, self-directed format.

The result, inevitably, was that we received more requests for builds than we anticipated, and we were unprepared to fulfill them. While we created regular builds of the game throughout development, their frequency and quality was insufficient to meet demand. This inhibited our ability to show off the game to the press when unexpected opportunities arose. When we did assemble builds on short notice, it slowed our momentum and caused a ripple effect on the rest of development.

team breakdown

- 6 full-time and 1 contract animator
- 10 full-time and 5 part-time artists
- 1 full-time and 1 part-time audio developer
- 2 full-time and 2 part-time designers
- 1 part-time and 4 contract concept artists
- 4 full-time and 2 part-time QA testers
- 1 full-time and 2 part-time producers
- 9 full-time and 1 contract programmer

The DEADLY CREATURES team.

Spending resources on builds is analogous to allocating a budget for marketing and insurance. It's tempting to cut back on these costs, but it can end up costing you more in the long run. That was certainly the case for us. We now believe that producing multiple, high-quality, self-directed builds throughout development would have actually reduced our costs, while also putting us in a better position to market our product internally and externally.

4) GRAY-BOXING. One of our development goals was to use gray-boxing to maximize iteration and minimize rework. During development, it felt like we were executing this goal successfully. In hindsight, we could have leveraged gray-boxing more often and more effectively.

We had used gray-boxing techniques on previous projects, but never to the extent that DEADLY CREATURES required. We were effective at creating gray-box content for environments, animations, cut scenes, and gameplay, but we struggled to create and evaluate all our gray-box content in the context of working levels with complete gameplay mechanics. Various constraints often led us to review gray-box content in isolation, and then move on. When we were able to review the content in the proper context, we received better critical feedback, which improved quality and reduced rework.

Most gray-boxing used in environments and cut scenes was created by an artist or animator under a designer's direction. The quality of this work was proportional to how effectively the designers and artists communicated with each other about abstract concepts.

In the future, we want to have designers provide the initial gray-box versions of their environments and cut scenes. We expect this will empower them to iterate quicker and generate results that more accurately reflect their vision.

5) UNDERSTANDING THE WII REMOTE. Leveraging the Wii remote to its full potential was a key design goal. However, we began preproduction before the Wii had been released, and our imaginations led us to devise overly ambitious control schemes. When we were finally able to prototype the control schemes on the Wii and play other Wii titles, we realized that our expectations were beyond the capabilities of the technology.

With the Wii remote in hand, we spent significant time trying to bend it to our will. After much experimentation, we concluded that complex gesture patterns were difficult to recognize with an acceptable level of accuracy. They also required significant design constraints, as recognizing such patterns required that we clearly identify the beginning and end of the gesture.

We eventually defined design constraints to help us avoid creating usage patterns that would punish a user for "mashing" gestures, and to avoid creating control mechanics that could be misinterpreted by our software. We limited the remote's gestures to cardinal directions—up, down, left, right, forward, backward—and the Wii nunchuk to non-directional shaking, as testing proved that users can make these gestures with a high degree of accuracy.

We then created usage scenarios that required a pattern of timed cardinal gestures with ample delay between each gesture. We also tried to map all cardinal directions to a valid input, so if players gesture-mashed, they would still get a satisfying experience during combat.

Reaching this point required significant iteration in design and implementation, which impeded the development of our gameplay and combat mechanics.

no arachnophobia

>> Our most significant successes came when we opened the floor to new ideas, and tackled risks head-on in novel ways. The game was a key achievement for our team, but also for our studio as a whole, since many of us at Rainbow have always wanted to expand beyond our core competency of making racing games. We are all very proud of DEADLY CREATURES and are confident that the risks we took in making it resulted in a fresh, surprising, and high-quality game that we hope our audience enjoys.

JAMES COMSTOCK *is a technical director at Rainbow/THQ. Email him at jcomstock@gdmag.com.*

GAME DATA

PUBLISHER
THQ

DEVELOPER
Rainbow

NUMBER OF FULL-TIME DEVELOPERS
35 at peak

LENGTH OF DEVELOPMENT
2 years

RELEASE DATE
February 9, 2009

HARDWARE
PCs with Intel Core 2 Duo 2.4ghz+, 3GB ram, Nvidia Geforce 7900+, RAID 0 hard drives

SOFTWARE
Visual Studio 2005, Metrowerks CodeWarrior, Perforce, 3ds Max 9, Maya 2008, Photoshop CS2, Flash, Adobe CS3, Cakewalk Sonar, Sony Sound Forge, Digidesign Pro Tools HD

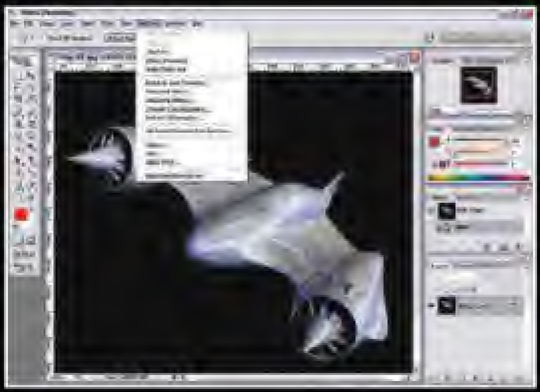
TECHNOLOGY
Bink, FMOD, Havok, Lua, Scaleform

PLATFORM
Nintendo Wii

Perforce | *Fast* Software Configuration Management

Introducing P4GT,

a productivity feature of Perforce SCM.



P4GT

The Perforce Plug-in for Graphical **Tools,** P4GT, makes version **control** painless by **seamlessly integrating Perforce** with **leading graphical tools. Drop-down menus allow access to Perforce from within 3ds Max, Maya, Softimage XSI, and Adobe Photoshop.**

Art and development teams can standardize on Perforce to version and manage both source code and digital assets. Enhanced collaboration during the design process helps teams to work together in real time to release small patches or create whole new worlds.

P4GT is just one of the many productivity tools that comes with the Perforce SCM System.

PERFORCE SOFTWARE

Download a free copy of Perforce, no questions asked, from www.perforce.com. Free technical support is available throughout your evaluation.

All trademarks and registered trademarks are property of their respective owners. Adobe screen shot reprinted with permission from Adobe Systems Incorporated.

Launch of Intel® Graphics Performance Analyzers: Virtual Graphics Support for Your Game Studio

You have an hour to kill before your next class. What to do? Study for your physics test? Nah, you whip out your lightweight laptop and continue the cool new game you were playing last night!

The shift from desktop PCs to laptops is creating new opportunities for game companies to expand their customer base. Gaming on the go—anytime, anywhere—is becoming commonplace. Game developers who once made games that were considered exclusively "high end" are now reaching a broader audience by making their games scale better across platforms. In 2008 mobile integrated chipset sales outsold discrete desktop graphics card sales for the first time in history. By 2013 mobile chipsets are expected to outsell discrete graphics cards by more than three to one.[1] As these numbers continue to grow, developers need tools to help them quickly optimize gameplay to reach this broader market.

One of the characters from Gas Powered Games new game *Demigod**.

New tools, such as the Intel® Graphics Performance Analyzers (Intel® GPA), are helping developers optimize game code for integrated graphics, so they can hit the performance targets needed while still delivering the great visuals their customers want.

Unlike some other tools on the market that were really intended for internal use and don't fit the needs of customers, Intel worked with a number of game companies to identify the features game developers needed most, and designed the tools specifically to fulfill those needs. "This is just the start," said Dave Shinsel, Intel GPA engineering manager. "We have a bunch of cool features we'll be implementing. We continue to listen to our customers; they tell us the features that are most important, and we build those first."

[1] McCarron, Dean, 2008, for Mercury Research. PC Graphics 2008; Updated Edition 4Q2008 Report. Available at www.mercuryresearch.com

Mark Randel, president and chief technology officer of Terminal Reality, said his team used Intel GPA extensively on their upcoming title *Ghostbusters*: The Video Game*. Based on the smash hit motion picture franchise, the new game reunites theoriginal cast members not only with voice work, but also within the story line. The game uses graphics to enhance both humor and fright and is meant to appeal to all key market segments.

Taking advantage of the opportunity to sell games into the mobile chipset space makes sense only if the effort doesn't substantially increase development time. Randel reported that his team was pleasantly surprised when Intel GPA helped make this a reality. "The Intel Graphics Performance Analyzers helped us mainly by reducing development time, since the tool provides a focus on specific problem areas. It also allowed our game to support the widest possible audience, by being able to run on Intel® Graphics. The benefit is mainly cost savings; the tool helps the engineer focus on the problem areas versus trial and error experiments. You can locate problem areas quicker than just turning visuals in the game on and off."The Intel GPA consists of two tools: the Frame Analyzer and the System Analyzer. Both tools sit on a network-based architecture, making data collection less intrusive and more suited to remote analysis than similar analyzers on the market. Because Intel GPA performs the calculations on a different machine than on the one running the game, developers don't have to worry about tool overhead.

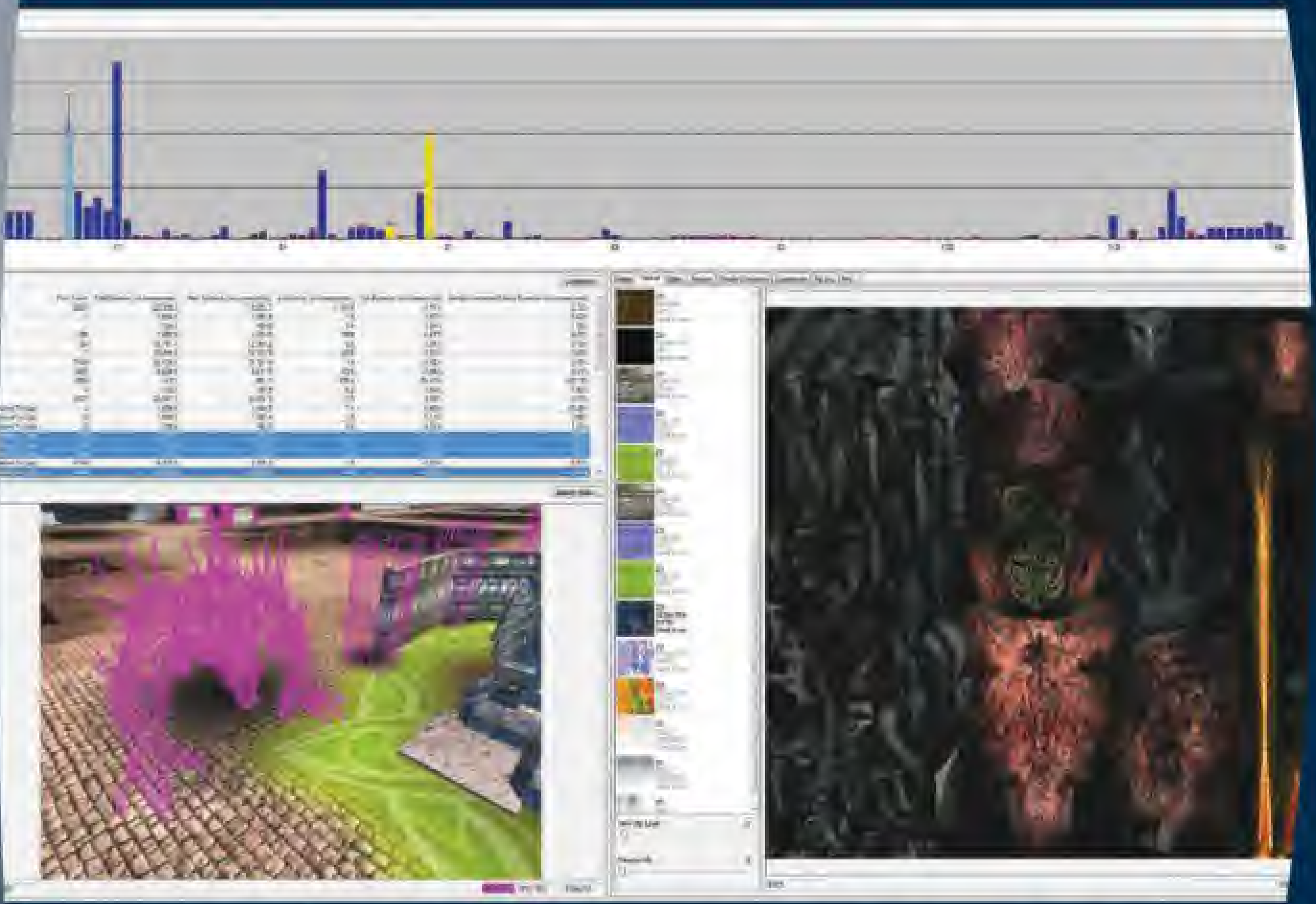


Figure 1. User interface of the Intel® Graphics Performance Analyzers' Frame Analyzer tool.

Early on, Randel knew he wanted to support more than just the high-end desktop space. "Intel GPA is a great first step into optimizing games for integrated graphics," he said. "With the client/server approach, you can measure the system in real time with minimal overhead to the target application. For example, you can launch your app, move to a problem area, and see where your time is going from buffer locks to state changes, and even down to the chip level if that is what you need to get it running faster."

The Frame Analyzer (Figure 1) allows developers to inspect and adjust graphics API-level interactions on a frame-by-frame basis. This capture- and playback-based tool shows detailed frame performance, with draw calls visualized on a GPU duration graph. Using the scene-overview feature, which gives a spreadsheet view of the same GPU information, developers can drill down from the full frame to single draw calls. Full-render state overrides, shader overrides, and other high-level experiments are supported with real-time feedback. For example a developer can modify a shader directly in the tool and immediately see if that change affected the frame time, region time, or draw-call time. Similarly, the developer can modify the DX state or run a high-level experiment, such as a simple pixel shader, and immediately see if that change affected the frame time, region time, or draw-call time.

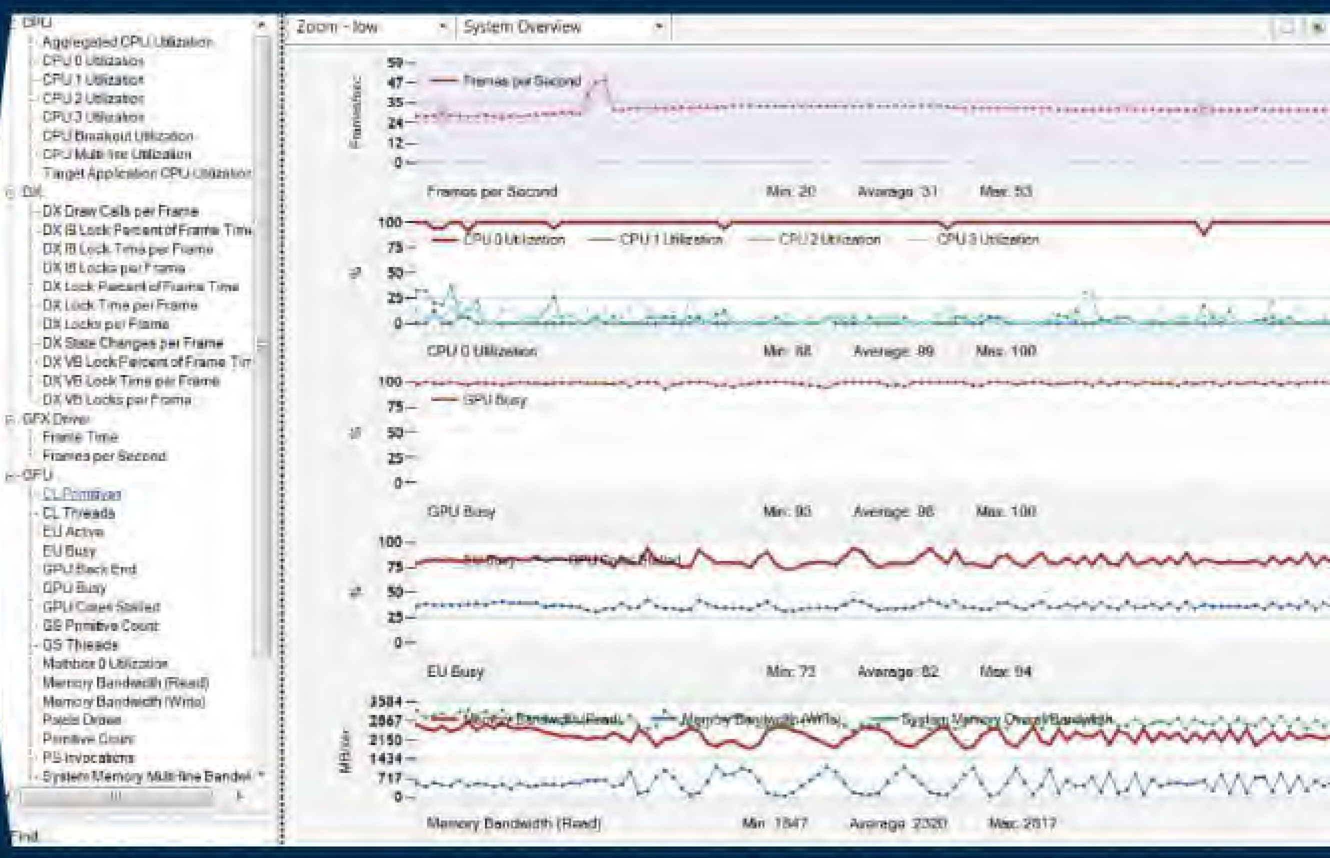


Figure 2. User interface of the Intel® Graphics Performance Analyzers' System Analyzer tool.

The SEGA development team working on *Empire*: Total War* used the Frame Analyzer extensively. Chris Southall, technical director at SEGA, said, "We have mainly used the frame level of Intel GPA. We have made changes to the level of detail in the game, made some art changes, optimized some shaders, changed draw order, and reduced overdraw. Using the system-level debugger, we have found a few bottlenecks with the way we were filling vertex buffers, which have also been fixed."

The System Analyzer (Figure 2) is a high-level, real-time performance tool with game pause-and-resume capabilities that provides a system-level footprint of game performance as well as a single-frame-capture button that transitions to the Frame Analyzer. Developers can experiment with Microsoft DirectX* state overrides, as well as customizable drag-and-drop metrics.

Bartosz Kijanka, vice president of engineering for Gas Powered Games, said his team relied on the Intel GPA tools for the company's forthcoming title: *Demigod**. A fast-paced, real-time strategy game, *Demigod* blends role-playing elements with tactical combat. Players can choose from a variety of wildly customizable demigods, who fight in ancient battle arenas for the right to ascend into the pantheon of gods.

Kijanka was intent on supporting integrated graphics chipsets from the start, so the Intel GPA tool was invaluable. "The System Analyzer helped us identify bottlenecks in the graphics pipeline through a few simple and easy-to-understand graphs," said Kijanka. "The Frame Analyzer is an amazing tool for drilling down deep into our engine's render pipeline and identifying individual pieces of work that may be disproportionately expensive. What is particularly impressive about this module is that the Frame Analyzer allows us to test possible fixes and to see the results immediately."

Using Intel GPA significantly reduced the cost of identifying and fixing graphics performance problems on the PC, according to Kijanka. "Many problems are costly to repair because they are very time consuming to diagnose," he explained. "They are time-consuming because they often require creation of project-specific instrumentation or project-specific tools. Spending less time on such tools allows us to spend more time on actually making the games, which is something every game developer wants to do."

The System Analyzer allows developers to run experiments that pinpoint common problems. The simple pixel-shader-override mode replaces every pixel shader with one that writes a constant color to the render target. A large increase in the frame rate after enabling this mode means that the game is spending a large proportion of time in either pixel shader compute or stall conditions.

The 1x1 scissor-override mode causes all pixels to be discarded after the pixel shader has run, before the pixel values are written to the render target. A significant increase in the frame rate after enabling this mode means the pixel-fill rate may be a potential bottleneck. Developers can then examine the operations, such as stencil and alpha blending, in the graphics pipeline to determine optimization potential.

Potential performance bottlenecks in the use of texture maps can be identified by using the 2x2 texture-override mode, in which all textures for a scene are replaced with a simple 2x2 pixel texture. If the override mode significantly improves the frame rate, the GPU may be bottlenecked on texture memory reads. If the total texture size is high for a scene, developers may want to consider reducing texture bandwidth in various ways.

The System Analyzer also collects various CPU and GPU metrics while the

application is running. Developers can analyze the results and perform various "what if" scenarios to help isolate performance bottlenecks. The metrics data are stored in a local database and can be displayed in chart form for interactive analysis. Developers can pause the application and perform on-the-fly modifications without changing the application code.

Intel's software team worked hard to create an intuitive interface for the tools. For example, the Frame Analyzer displays all DX data in visual form when possible. This includes the render targets, draw-call highlighting for the draw-call selection set, textures, and so on. The visual element of the tool suite allows any game developer to pick up the tool and use it effectively immediately.

"PC developers have never had such a reliable, flexible, and extensive tool set for identifying and resolving complex graphics performance issues," said Kijanka. "Intel GPA collects and presents a breadth and depth of information about graphics performance that we have never previously seen for the PC platform. Our games perform much better, across a wider range of hardware than ever before, including many graphics parts with high market penetration that we have traditionally not had the resources to optimize for. This means more customers can play our games and have a great, high-frame-rate experience, even on older or lower-cost PCs, and we definitely benefit from that."

For more details, go to:
www.intel.com/software/gpa •

To get more great articles like this one, subscribe today to Intel® Software Dispatch for Visual Computing at:
www.intelsoftwaregraphics.com

VISUAL ADRENALINE intel Software

Intel does not make any representations or warranties whatsoever regarding quality, reliability, functionality, or compatibility of third-party vendors and their devices. All products, dates, and plans are based on current expectations and subject to change without notice. Intel, Intel logo, and Intel Core are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others. | Copyright ©2009, Intel Corporation. All rights reserved. 04/09/SM/CS

REPORT FROM THE SHOW FLOOR

GAME DEVELOPERS CONFERENCE 2009

ENABLING SMALL TEAMS WAS THE UNDERLYING MESSAGE OF MANY OF THE GDC 09 EXHIBITORS. FROM PROTOTYPING TO FULL-SCALE DEVELOPMENT, HERE IS A CROSS SECTION OF SOME OF THE TOOLS THAT ALLOW DEVELOPERS TO DO MORE WITH LESS. —*Jeffrey Fleming*

VIRTOOLS 5
3DVIA
http://a2.media.3ds.com

The Virtools game prototyping and game production engine has been updated to version 5. The engine now features Lua integration and blend shape support for fast integration of art and animation assets. The Virtools Scripting Language also adds Lua support. Games created with the engine can be deployed on Xbox 360, Nintendo Wii, PC, Mac, or the Web.

SOUNDSEED AIR
AUDIOKINETIC
www.audiokinetic.com

Audiokinetic is continuing to expand on its SoundSeed family of sound generators for Wwise. Taking a synthesis-based approach, SoundSeed promises to enable continually variable sound effects while reducing file sizes to a minimum. SoundSeed Impact is already available and is focused on generating short transient impact sounds such as clangs, thunks, and crunching noises.

At GDC Audiokinetic was showing the upcoming SoundSeed Air, which models a variety of wind effects like the sweep of an arrow or the thwip of a passing bullet. It is also able to realistically model ambient wind effects and gives audio designers full control over wind speed, direction, and sound reflectors in the environment. SoundSeed Air uses parametric synthesis to create its wind effects in real-time and requires no source files. Audiokinetic plans to make SoundSeed Air available later this year.

SUBSTANCE AIR
ALLEGORITHMIC
www.allegorithmic.com

Replacing traditional bitmaps with procedural texture generation can substantially reduce file sizes as well as open up new creative opportunities. Currently under closed beta testing, Allegorithmic's Substance Air is the latest iteration in the company's line of texturing middleware. Utilizing both an editor and a runtime engine, Substance Air allows artists to freely mix SVGs and bitmaps with procedural textures as well as create dynamic assets that can be altered by end users or by the game engine.

"There is a direct relationship between the size of a game and the money it will generate in the end. Especially in the case of free-to-play games," Sébastien Deguy, ceo of Allegorithmic told us. "Gravity is one of our clients, and by reducing the size of the game textures, they've seen the number of players actually entering the game after they registered on the web site go from 16 percent to 26 percent. That means they've seen less people abort downloading a game because it was taking too long. Players want to click and play immediately." Allegorithmic plans to make Substance Air publicly available within several weeks.

AUTODESK KYNAPSE 6
AUTODESK
www.autodesk.com

Autodesk gave GDC attendees an early look at their upcoming release of Kynapse 6. "It doesn't do decision making for you. We believe that is very integral to the game play and we leave that to the game designers and programmers to decide how it's implemented. What Kynapse does is spatial awareness, so that a character is fully aware of a 3D environment with true 3D dynamic pathfinding and team coordination," Leonard Teo of Autodesk told us.

The AI tool features new "flat" pathfinding technology that enables simplified data generation workflows, flexible runtime data streaming, and the adding of new paths at runtime. Authoring is made easier by Kynapse's new remote debugging tool that enables game variables to be inspected with an interactive 3D view. The debugging tool also allows recorded sequences to be played back with full access to the data in the recorded files.

Kynapse's hierarchal 3D pathfinding gives NPCs the ability to plan paths for maps that extend beyond what is loaded into memory. Using a level of detail method, paths are computed at a low level and then refined as more data becomes available. The company also announced that it is working on integration between Kynapse and NaturalMotion's morpheme animation middleware.

GAMEBRYO LIGHTSPEED
EMERGENT
www.emergent.net

Emergent is responding to the increasingly fluid nature of game creation with the release of Gamebryo LightSpeed. Using a development framework built on Gamebryo technology, LightSpeed features a new data-driven entity and behavior architecture that enables changes to game assets to be reflected in real time without the need for recompiling.

Montgomery Markland of Killswitch Entertainment spoke with us and described using LightSpeed to prototype his studio's new project, BEAT, in eight days. "LightSpeed lets you rapidly prototype and rapidly iterate. The ability to do that reduces your risk because you can pitch a lot easier. You're not just pitching on paper, you can show people something that they can grab on to and understand and see the aesthetics of," he said.

The LightSpeed Toolbench IDE is designed to host all tools as plug-ins, enabling developers to use pre-integrated tools out of the box and facilitating easy integration of custom-built tools. It also supports a number of scripting systems and includes a full-featured Lua debugger tool as well. LightSpeed's Entity Modeling Tool enables the easy creation of new data-driven game objects by mixing parent entities. WorldBuilder is a level editor integrated into the Entity Modeling Tool that allows developers to model entities, define their properties and behaviors, and then see the results reflected immediately in the game world.

LightSpeed also includes drag and drop positioning of objects, exporters for industry standard DCC tools, built-in support for PhysX, as well as support for the variety of Emergent Partner technologies that are available.

"The most pernicious problem that faces content generators is that pipe between the tools and the platform. That is the biggest bottleneck on every project and LightSpeed completely solves that problem. It totally levels the playing field. In the past it was like this guy is playing a guitar and this guy is driving a bulldozer. Now everybody is playing instruments on the team. Everybody is collaborating with the same tools, with the same language. It means the producer, the designer, the artist, and the programmer can all work based off the same platform. It's almost like it gives us a real-time, straight to platform whiteboard," Markland said.

BLADE 3D GAME ENGINE

BLADE3D

www.bladegamesworld.com

Blade3D says its game creation platform is a fast, artist-friendly engine for creating 3D games for Windows XP, Vista, and Xbox 360 (requires XNA Creators Club Membership for Xbox 360 deployment). Visual programming tools coupled with Blade3D's real-time design environment allow for changes in editors to be reflected instantly in the game world. Game logic can be authored using C# based scripting or a graphical programming language called Visual Logic Diagram. Blade has also created a marketplace for creators to buy and sell game assets for Blade3D. Blade's marketplace includes models from the Daz 3D catalog and is also positioned with Vyk Games in Shanghai to provide full outsourcing services.

"The way we've structured our business, we want to be there for the developer, wherever they are. So if they're a hobbyist, or they have an idea for a game but they're not an artist, they can go to our marketplace. If they're an indie guy and have some programmers but need an engine, there's Blade. If they're a professional developer and have a big project but the cost of the production is going to be too high, we've got the services component. We want to scale it depending on who needs what and make it affordable," Tony Garcia, then CEO of Blade 3D told us.

HEROENGINE 2009 VOLUME 1

SIMUTRONICS

www.heroengine.com

For years Simutronics has been using a decentralized model for developing and managing their MMOs by giving their GameMasters the ability to work offsite and collaborate over the network to provide content and resolve support issues. The company has embedded this methodology into its HeroEngine development platform that allows teams to work concurrently from multiple locations on a shared game world that is running in real-time. Game assets are drawn from a central repository, and HeroEngine's data object model enables definitions that are changed in realtime to be replicated system-wide without recompiling. Content in a running game can also be updated without the need to bring servers down.

"Our database is morphing in real-time so we don't ever have to shut anything down or manage migrations of databases between versions. That's a lot of friction to pull out of the development process," David Whatley, president of Simutronics told us.

HeroEngine can handle 100,000 players in a single shard, the company claims, and its Seamless World technology allows developers to link together unlimited virtual areas of any size. The HeroEngine backend supports both Windows and Linux servers.

"MMOs cost way too much time and money to build, taking four years and tens of millions of dollars," Whatley said. "One of the reasons that there's not a lot of innovation in the MMO space is because you can't afford to take much risk under that scenario. Going forward, this kind of tool is going to be great for allowing people to build out a small amount of content with some innovative features, put it in to the marketplace, see if it catches on, and then if it does, start iterating really fast on content and pushing it out into the environment."

HAVOK AI

HAVOK

www.havok.com

Havok stepped in to the AI arena with the announcement of their pathfinding solution at GDC. Integrating with Havok's Physics, Destruction, Animation, and Behavior products, Havok AI enables dynamic pathfinding and is designed to quickly generate an optimized nav mesh. Havok claims that the AI's dynamic pathfinding can track thousands of moving obstacles in real-time. It also includes a predictive local steering module that allows characters to react to moving obstacles and avoid traffic jams. The software is fully extensible and customizable as well as being multithreaded and platform optimized.

IISU 3D GESTURE RECOGNITION SDK

SOFTKINETIC

www.softkinetic.net

Softkinetic's iisu platform provides tools and APIs for developers looking integrate 3D gesture recognition into their games. Working with infrared depth sensing cameras, iisu supports automated camera configuration and can classify and filter data coming from the camera. The software is able to identify and track individual body parts allowing for recognition of hand and finger gestures or full-body movement. The SDK promises to speed development by allowing creators to focus on gameplay rather than gesture recognition algorithms.

BLUE MARS SOFTWARE DEVELOPMENT KIT

AVATAR REALITY

www.avatar-reality.com

The Blue Mars SDK is now available at no cost to approved third party developers who want to begin work on Avatar Reality's virtual world platform. Built on CryENGINE 2 technology, Blue Mars uses a secure transaction backend that allows game developers to support a variety of business models including, free-to-play, subscription, and micro-transactions.

It also supports scalable attractions for simultaneous users, which the company says should eliminate customer restriction limits. Aiding development for Blue Mars are LUA support and a casual games API, as well as an asset pipeline that supports industry standard DCC tools and Flash-based user interfaces. Avatar Reality intends to have a limited public beta of Blue Mars available by June of this year.

XAITENGINE

XAITMENT

www.xaitment.com

Xaitment has created a suite of modular AI tools that target specific AI needs, allowing developers to mix and match according to their project's requirements. For pathfinding solutions, the company offers xaitMap for creating navigation maps, and xaitMove2 for authoring bot behavior. xaitControl can be used to create state machine behavior while xaitKnow and xaitThink provide high-level AI functions such as world knowledge and rule-based autonomous decisions. Xaitment is bundling xaitMap, xaitMap, and xaitMove2 together under an initial no cost license that allows start-up projects to move to a full license once the game has been sold to a publisher.

Avatar Reality's Blue Mars SDK is now available.

VANCOUVER, CANADA

GAME DEVELOPMENT DIVERSITY

THE PACIFIC RIM IS HOME to a number of unique game industry clusters and Vancouver in British Columbia joins California and Washington as a major center of gravity for development talent. For almost three decades the Vancouver scene has been dominated by Electronic Art's massive EA Canada facility and Radical Entertainment's equally productive studio.

However, in recent years the city has supported a healthy ecology in which the big studios have acted as proving grounds for ambitious developers who have gone on to create numerous start-up companies of their own.

DISTINCTIVE SOFTWARE

>> Much of Vancouver's current development landscape is fed by streams of influence that stretch back to 1982 and the founding of Distinctive Software. Distinctive was formed by Don Mattrick and Jeff Sember, and was closely associated with the publisher Accolade during its early years. Initially working on a number of PC ports and action games, the studio soon distinguished itself by creating the first entry in the long-running TEST DRIVE series in 1987. Over the next several years Distinctive continued to hone its racing game expertise with the subsequent releases of THE DUEL: TEST DRIVE II, GRAND PRIX CIRCUIT, and STUNTS.

In 1991 Distinctive joined Electronic Arts, becoming the publisher's first studio acquisition. Renamed EA Canada, the studio was a cornerstone of EA's rapidly growing dominance in sports games. In 1994 EA Canada partnered with *Road & Track* magazine to produce THE NEED FOR SPEED for Trip Hawkins' 3DO dream machine. The game took advantage of the new hardware, to raise the bar for future racing sims by including realistic car handling and careful attention to engine sounds, along with race commentary and video clips of its exotic cars in action.

In addition to the NEED FOR SPEED series, EA Canada has gone on to produce a variety of sports titles for Electronic Arts including SSX, the NBA, NFL, and FIFA STREET series, among others. After leading EA Canada, Mattrick went on to become the president of EA's worldwide studios before moving on to Microsoft in 2007.

Veterans of EA Canada have formed the basis of many smaller Vancouver area studios. Members of Propaganda, the creators of the new TUROK, accrued years of experience at EA Canada before starting their own studio. The staff at Deep Fried Entertainment, A.C.R.O.N.Y.M. Games, Jet Black Games, and Koolhaus Games, have all spent time at EA Canada as well.

RADICAL ENTERTAINMENT

>> Radical Entertainment, creator of the upcoming PROTOTYPE, has a long history in Vancouver game development. Since the studio's founding in 1991, Radical's strategy of developing sure-fire licensed titles such as SIMPSONS ROAD RAGE and SCARFACE: THE WORLD IS YOURS, along with original IP like DARK SUMMIT has enabled it to grow into a major employer of Vancouver game development talent.

Rockstar Vancouver, maker of BULLY, began life as Barking Dog Studios in 1998. Founded by developers from Radical, Barking Dog created several games including HOMEWORLD: CATACLYSM and GLOBAL OPERATIONS. Take-Two Interactive bought the studio and brought it into the Rockstar family in 2002. The studio is currently at work on MAX PAYNE 3.

Another Radical alumnus, Martin Sikes, left in 1998 to start Black Box Games. Four years later the studio joined EA Canada and created NEED FOR SPEED: HOT PURSUIT 2. Black Box took the series in new directions with NEED FOR SPEED UNDERGROUND, NEED FOR SPEED CARBON, and NEED FOR SPEED PROSTREET. Spun off as an independent EA studio in 2005, Black Box gave the skateboarding genre a much-needed shot in the arm with the release of SKATE three years later. Although Black Box remains an individual entity within EA, the studio has since been moved back into EA Canada's facilities in a cost saving effort. Sikes went on to help form United Front Games in Vancouver before passing away in 2007.

RELIC

>> Specialists in real-time strategy, Relic made a strong debut in 1999 with the release of HOMEWORLD. The space combat game took the somewhat moribund genre in creative new directions with a true 3D environment and a minimalist, hard-science aesthetic that emphasized fire and movement over resource farming. HOMEWORLD also featured a sophisticated visual design that harkened to the great work done by European science fiction illustrators of the 1970s.

Following their acquisition by THQ in 2004, Relic released WARHAMMER 40,000: DAWN OF WAR and two years later brought the critical smash COMPANY OF HEROES to market. Former Relic staff members have since split off to form Smoking Gun Interactive and are currently at work on an unannounced project.

LITTLE STUDIOS, BIG PROJECTS

>> In addition to the major studios of Vancouver, a plethora of smaller studios have found a home in the area. While they may not have the three and four digit staff numbers of the big studios, their projects are no less impressive.

Hothead Games is reinvigorating the adventure game genre with the episodic PENNY ARCADE ADVENTURES: ON THE RAIN-SLICK PRECIPICE OF DARKNESS. The studio is also at work on Ron Gilbert's upcoming DEATHSPANK. Ironclad Games, the creator of the real-time strategy game SINS OF A SOLAR EMPIRE, is located in nearby Burnaby. Blue Castle Games, which developed the sports titles FRONT OFFICE MANAGER and THE BIGS, is currently at work on DEAD RISING 2 for Capcom. As part of the Foundation 9 family of studios, Backbone Entertainment maintains a studio in Vancouver. Next Level Games created SUPER MARIO STRIKERS and MARIO STRIKERS CHARGED for Nintendo. Slant Six Games has taken over the SOCOM franchise for Sony and is currently at work on SOCOM: U.S. NAVY SEALS FIRETEAM BRAVO 3. Threewave Software began by creating a Capture the Flag mod for QUAKE and has since provided multiplayer content for a variety of high-profile shooters including RETURN TO CASTLE WOLFENSTEIN, DOOM III: RESURRECTION OF EVIL, and ARMY OF TWO.

By supporting such a range of development activity, both on a large and small scale, Vancouver is well positioned to influence the game industry throughout the 21st century.

JEFFREY FLEMING *is production editor of* Game Developer. *He likes William Gibson books. Email him at jfleming@gdmag.com.*

NITTY GRITTY UNIT TESTING

ALL YOU NEED TO KNOW TO START UNIT TESTING YOUR CURRENT PRODUCT

IT'S ONE THING TO SEE A PERSON DRIVE a manual transmission car and have a theoretical understanding of what the pedals do and how to change gears. It's another thing to drive on the street, safely and without stalling. There are certain activities that, to execute successfully, a person needs to have not only knowledge of all the details that go into making them happen, but also some hands-on experience. Unit testing is one of them.

The good news is unit testing is a lot simpler than driving stick. But there are a lot of small details that game programmers need to get right to do it successfully.

Even after reading about unit testing and being convinced of its benefits, programmers are not always sure how to get started. It's not my intention to convince you of the many benefits of unit testing (I hope you're already convinced), but rather, to describe some very concrete tips to help you get your hands dirty right away.

ILLUSTRATION BY JONATHAN KIM

GOALS OF UNIT TESTING

>> Unit tests are used for many kinds of testing:

- CORRECTNESS TESTING: to check that the code behaves as designed.
- BOUNDARY TESTING: to check that the code behaves correctly in odd or boundary situations.
- REGRESSION TESTING: to check that the behavior of the code doesn't change unintentionally over time.
- PERFORMANCE TESTING: to check that the program meets certain minimum performance or memory constraints.
- PLATFORM TESTING: to check that the code behaves the same across multiple platforms.
- DESIGN: tests provide a way to advance the code design and architecture (usually referred as test-driven development or TDD).
- FULL GAME OR TOOLS TESTING: technically this is a functional test, not a unit test, because it involves the whole program instead of a small subset of the code, but a lot of the same techniques apply.

Some developers use unit tests only for one of the reasons listed above, while others use many kinds of tests for a variety of reasons. It's important to recognize that because there are so many different uses for unit tests, no single solution is going to fit everybody. The ideal setup for some of those situations is going to be slightly different than for others, but the basics are the same for all of them.

When working with unit tests, the main goals are:

- **Spend as little time as possible writing a new test,**
- **Be notified of failing tests, and see at a glance which ones failed and why,**
- **Trust the tests (have them be consistent from run to run and robust in the face of bad code), and**
- **Create a testing framework.**

TESTING FRAMEWORK

>> Most of us have created one-off programs in the past to test some particularly complicated code. It's usually a quick command line program that runs through a bunch of cases and asserts that the results were correct for each one. That's the most barebones way of creating unit tests.

Unfortunately, it's also a pain, and it misses on most of the unit testing goals described in the previous section. Creating a new program just to run a new set of unit tests is a nuisance, we have to go out of our way to run the

tests, and it's usually out of date faster than the latest Internet meme. Perhaps this explains in part why a lot of programmers have an initial aversion to writing unit tests at all.

If you're considering writing even a small unit test, you should use a unit-testing framework. A unit-testing framework removes all the busywork from writing unit tests and lets you spend your time on the logic of what to test. This doesn't mean that the framework writes the tests for you—be wary of any tool that claims to do that! Rather, a unit-testing framework is simply a small library that provides all the glue for running unit tests and reporting the results. However, you still need to use your brain and do (some of) the typing. Sorry.

A quick search will reveal plenty of unit testing frameworks in your language of choice to choose from, most of which are free and open source so you can rely on them and modify them to suit your needs.

For C/C++ and game development, I strongly recommend starting with UnitTest++. Charles Nicholson and I wrote that framework a few years ago specifically with games and consoles in mind. Many game teams have adopted it for their games and tools, and it has been used on a lot of different platforms, including current and last generation consoles, Windows, Linux, Mac, and the iPhone. In most situations, it's a straight drop-in to the project to get you up and running.

If you end up using a different testing framework, or if you code your own, the techniques described here still apply, even if the syntax is slightly different.

HELLO TESTS

>> Writing your first test is easy as pie. Try this sample code:

```
#include <UnitTest++.h>

TEST(MyFirstTest)
{
    int a = 4;
    CHECK(a == 4);
}
```

To run it, you need to add the following line to your executable somewhere (we'll talk more about the physical organization of tests in a moment):

```
int failedCount =
UnitTest::RunAllTests();
```

Done. Easy, right?

When you compile and run the program, you should see the following output:

```
Success: 1 test passed.
Test time: 0.00 seconds.
```

Let's add a failing test:

```
TEST(MyFailingTest)
{
    int a = 5;
    CHECK(a == 4);
}
```

Now we get:

```
/fullpath/filename.
cpp:17: error: Failure in
MyFailingTest: a == 4
FAILURE: 1 out of 2 tests
failed (1 failures).
Test time: 0.00 seconds.
```

That's great, but if we're going to diagnose the problem, we need to know the value of the variable `a`, and all the test is telling us is that it's not 4. We can change the `CHECK` statement to the following:

```
TEST(MyFailingTest)
{
    int a = 5;
    CHECK_EQUAL(4, a);
}
```

And now the output will be:

```
/fullpath/filename.
cpp:17: error: Failure in
MyFailingTest: Expected 4
but was 5
FAILURE: 1 out of 2 tests
failed (1 failures).
Test time: 0.00 seconds.
```

That's much better. Now we see both the error information and the value of the variable. Virtually all unit testing frameworks include different types of `CHECK` statements to get more information when testing floats, arrays, or other data types. You can even make your own `CHECK` statement for your own common data types, such as colors or lists.

As a bonus, if you're using an IDE, clicking on the test failure message should bring you automatically to the failing test statement.

WHEN TO RUN

>> When to run unit tests will depend on what's being tested and how long it takes to test it. In general, the more frequently you run the tests, the better. The sooner you get feedback that something went wrong (maybe before code is checked in and is disseminated to the rest of the team), the easier it will be to fix. On the flip side, realistically, building and running a set of unit tests takes a certain amount of time, so it's important to find the right balance between feedback frequency and time spent waiting for tests.

At the very least, all tests should run once a day during the nightly build process in your build server. (You have a build server, don't you? If not, stop right here and read The Inner Product column,"The Heartbeat of the Project," in the August 2008 issue before you continue.) It doesn't matter how long they take or how many different projects you need to run. Just add the tests to the build script, and hook their output into the build results.

On the other extreme, you can build your tests every time you build the project and execute them as a post-build step. That way, any time you make a change to a project, all tests will execute and you'll see if anything went wrong. This approach works great, but I wouldn't recommend using it if the tests add more than a couple of seconds to the incremental build time; otherwise, they'll be slowing you down more than they help.

Most developers will find that the approach that makes the most sense for them is somewhere between these two extremes, for example, taking a small and fast subset of tests that are more likely to break, and running those with every build. Whenever code is checked in to your version control system, the build server can run those tests, plus a few others that are slower. And finally, at night, you can bring out the big guns and run those really long and thorough tests that take a few hours to complete.

One way to lower the impact of constantly running tests is to separate those tests into different projects, or, if your framework supports them, into different test suites, which allows you to decide which sets of tests to execute at runtime.

REPORTING RESULTS

>> If a unit test fails and nobody notices, is it really an error?

Just running tests isn't good enough. We have to make sure that someone sees failures when they occur and fixes the problems.

Most unit-testing frameworks will let you customize how the failure errors are reported. By default, they will probably be sent to `stdout`, but you can easily customize the framework to send them to debug log streams, save them to a file, or upload them to a server.

Even more important than seeing the actual error messages is automatically detecting whether there were any failures. After running all the tests, there is usually some way to detect how many tests failed. The program that was running the tests can detect failures, print an error message, and exit with an error code. That error code will propagate to the build server and trigger a build failure. Hopefully by now alarms are ringing across the office, and someone is on his way to fix the problem.

PROJECT ORGANIZATION

>> When people start down the unit test path, they often struggle to figure out how to physically lay out the unit tests. In the end, it really doesn't matter how you do it too much as long as 1) it makes sense to you, 2) the final build doesn't contain any tests, and 3) the unit tests are easy to build and run.

My personal preference is to keep unit tests separate from the rest of the code. Usually, I end up creating one file of tests for every `cpp` file, for example, `FirstPersonCameraController.h` and `.cpp` should have a corresponding `TestFirstPersonCameraController.cpp`. Since I use this convention regularly throughout all my code, I have a custom IDE macro to toggle between a file and its corresponding test file. I also put all the tests in a separate subdirectory to keep them as physically separate as possible.

I prefer to break up my code into several static libraries for each major subsystem: graphics, networking, physics, animations, etc. Each of those libraries has a set of unit tests, but instead of compiling them into the library, I make a separate project that creates a simple executable program. That project contains all the unit tests and links against the library itself, and in its main entry point, it calls the function to run all unit tests and returns the number of failures. This way, tests stay separate from the library, but they're still very easy to build.

If all your code is organized into libraries, and your game is just a collection of libraries linked together, that's all you need. But most games and tools have a fair amount of code that you might want to test in the project itself. Since the game is an executable, you can't easily link against it from a different project like we did before. In this case, I build the unit tests into the game itself, and I optionally call them whenever a particular command-line parameter like `-runtests` is present. Just be sure to `#ifdef` out all the tests in the final build.

MULTIPLATFORM TESTING

>> Running the tests on the same PC that you built the code on is straightforward. But unless you're only creating games and tools for that platform, you will definitely want to run your tests on different platforms as well.

Unit tests are an invaluable tool for catching slight platform inconsistencies caused by different compilers, architecture idiosyncrasies, or varying floating point rules.

Unfortunately, running unit tests on a platform other than your build machine is usually a bit more involved and not nearly as fast as doing it locally. Start by compiling the tests for the target platform. This is usually not a problem since you're already building all your code for that platform, and hopefully your unit testing framework already supports it. Next, upload your executable with the tests and any other data required to the target platform and run it there. Finally, get the return code back to detect if there were any failures.

Surprisingly, that final step is often the trickiest part of the process on a lot of console development kits. If getting the exit code is not a possibility, you need to get creative by parsing the output channel, or even waiting for a notification on a particular network port.

Some target platforms are more limited than others in both resources and C++ support. One reason UnitTest++ is a good choice for games is that it requires minimal C++ features (no STL), and it can be trimmed down even further (no exceptions).

For example, on past projects, running tests on PlayStation 3 SPUs was extremely useful, but required stripping down the framework to the minimum number of features. It also took some finagling to fit the library code plus all the tests into the small amount of memory available. We ended up changing the build rules for the SPUs so each test file created its own SPU executable (or module). We then wrote a simple main SPU program that could load each module separately, run its tests, keep track of all the stats, and finally report them.

Running a set of unit tests on the local machine can be an almost instant process, but running them on a remote machine is usually much slower and can take up to 20 seconds just with the overhead of copying them and launching the program remotely. For this reason, most developers will prefer to run tests on other platforms less frequently.

NO LEAKING ALLOWED

>> Finally, if you're going to have all this unit testing code running on a regular basis, you might as well get as much information out of it as possible. I personally find it invaluable to keep track of memory leaks around the unit test code. You'll have to hook into your own memory manager or use the platform-specific memory tracking functions. The basic idea is to get one memory status before running the tests and another one afterward. If there are any extra memory allocations, you've probably got a leak. In that case, you can report it as a failed build by returning the correct error code.

Watch out for static variables or singletons that allocate memory the first time they're used. They might be reported as memory leaks even though it wasn't what you were hoping to catch. In that case, you can explicitly initialize and destroy all singletons, or better, not use them at all and keep your memory leak report clean.

You're now armed with all you need to know to set up unit tests into your project and build pipeline. Grab a testing framework, and get your feet wet today.

NOEL LLOPIS *has been making games for just about every major platform in the last ten years. He's now going retro and spends his days doing iPhone development from local coffee shops. Email him at nllopis@gdmag.com.*

resources

UnitTest++ framework
http://unittest-cpp.sourceforge.net

size matters

www.rtpatch.com

RTPatch and Pocket Soft are registered trademarks of Pocket Soft, Inc.

NOKIA
Connecting People

This year at GDC, Tero Ojanperä gave an insight into what the world's largest mobile device manufacturer sees on the horizon. During his keynote at GDC 2009, Tero talked about Nokia's mobile gaming efforts, including N-Gage, and shared trends and lessons learned from the first year of the service.

Looking to the Future

With more mobile platforms in the market, developers now have more channels than ever before to reach users with great games experiences. Tero looked to the future consumption of digital entertainment via the mobile phone, revealing some impressive numbers. For example, emerging markets will account for 40% of mobile gaming revenue by 2012 and in developed markets, games continue to be the most popular 3rd party applications installed on devices.

Growing Mobile Gaming

This magnitude of growth has significant implications on mobile game development and represents a valuable opportunity for both game developers and Nokia. Tero Ojanperä explored some of the newest innovations in game development, from recent Nokia Research Center device technologies to the combination of Nokia services such as music and games in Dance Fabulous.

Publish to Ovi

He also emphasized the importance of social networking with the recent announcement of the Ovi Store and the launch of Publish to Ovi. Ovi Store is a next generation media service that will serve up content (including games) pro-actively based on the users' location and consumption patterns between friends. To access this distribution channel, Publish to Ovi lets content owners easily upload and monetize their content, making it available to millions of Nokia devices.

Bringing Meaning to Mobile Entertainment

Tero Ojanperä at GDC 2009

With mobile gaming capturing the imagination of consumers and developers alike, there is much in store for the future of games on the mobile phone.

To view full Nokia presentations from GDC 2009 including Tero Ojanperä's keynote, please visit www.insider.n-gage.com.

insider.n-gage.com

© 2009 Nokia

N-GAGE by Nokia ovi

IS 40 THE NEW 30?

THINKING ABOUT YOUR CAREER IN THE LONG TERM

MANY MOONS AGO, ALMOST FIVE YEARS IN FACT (or as we count in the game industry, "two product cycles ago"), I wrote a column on age discrimination called, "Never Hire Anyone Over 30" (August 2004). With game developers more focused on their careers these days in a tremulous global economy, it seemed like a good idea to revisit the topic and see what, if anything, has changed for the grizzled ancients of the game art world.

"Never Hire Anyone Over 30" was one of the most commented on and controversial columns I've ever written. Dozens of people wrote me to share their personal experiences about being on the wrong end of their third decade. The general consensus among the angry respondents seemed to be that the article was too tame.

A lot of folks in their mid-to- late 30s felt the business had no use for them and that no one was shy about showing them the door. Bitter veterans at 33 years of age, many artists recounted being slighted in favor of kids who could work all night, didn't have to answer to angry spouses at crunch time, and didn't dare talk back to management. Add in the dire effects that crunch culture can have on families (and on the health of sedentary folks in their 30s and 40s), and it seemed like the average career of a game artist was destined to last only seven or eight years.

Five years on, the game industry remains a young person's playing field, but the culture has shifted noticeably. Folks in their 30s are no longer quite the demographic rarities they once were. As a matter of fact, the IGDA's Diversity study claims the average game developer is now between 31 and 35. It wasn't long ago that the stereotypical developer was supposed to be a boomerang kid living in his parents' basement, surrounded by mountains of unwashed laundry and empty pizza boxes. Nowadays, an employed game developer is more likely to be a parent than to live with one, and the unwashed laundry pile probably includes some bibs and onesies. Only the pizza boxes remain true to form.

BRINGING PEOPLE TOGETHER

>> The age increase of our demographic reflects the maturation of the business as a whole. According to the Game Developer Research *Developer's Census 2008* there are now more than 40,000 developers in North America alone—and that's just the folks in core game development, not counting the retail and distribution sides of the industry. The pool of veterans must be a lot bigger now than five years ago by a fairly reasonable arithmetic assumption. But more importantly, the incredible increase in the scale of the industry has made a lot of headroom that wasn't here five years ago.

One of the most common reasons artists used to cite for exiting the game industry young was topping out, hitting a point in one's career where there was no obvious path to more responsibility, more creative freedom, or more money.

In the 2004 article, I wrote: "It might be easier to seduce us into management if there were management niches to fill—but in most studios, the career ladder is extremely short. You can become an art lead, and then an art director ... and then you're done and can retire. Without a lot of intermediate positions, it's tough to even know if you have the aptitude or the desire to lead a team."

In the intervening five years, the boondoggle known as next-gen development has expanded team sizes enormously. Next-gen bloat has been tough on the industry's finances and creative freedoms, but it's been a boon to middle management, and also, indirectly, to the careers of many artists who might otherwise have found their career paths stunted by the lack of ladders to climb.

BRINGING UP BABY

>> When I first discussed ageism five years ago, a lot of artists weren't sticking around long enough to mature into managers due to the punishing cycle of crunches, which often prevented them from pursuing a real family life.

Crunch certainly has not disappeared, but the flowering of new kinds of game studios and companies has definitely allowed many developers to stay in the game business while sidestepping the traditional studio-to-Best Buy model of development and distribution. Games for casual play, mobile devices, PCs, and download—even games for Facebook and MySpace—have forged new and typically smaller environments that beckon to folks who love games but can't reconcile the demands of a deadline-driven AAA studio with the needs of family life. It's a lot easier to approximate a traditional 9-to-5 work schedule when a company isn't looking for a product to hit store shelves to the tune of a $15 million marketing blitz and a raft of action figure tie-ins by a certain date.

Many senior artists also find a lot of satisfaction in putting their personal stamp on a smaller project, instead of toiling anonymously in the bowels of a 200-person team with a focus group driven design. Old school skills, like low-poly modeling, texture palette management, and limited frame rate animation, are still very much in demand in the casual and handheld spaces. Plus, middle-aged game artists are often grateful for the opportunity to share their work guiltlessly with their own children, instead of negotiating delicately between their duties as parents, their pride as creators, and, let's face it, the dirty looks they can expect from other parents if they introduce little Johnny's third grade classmates to their fine work on the chainsaw bayonet blood-spatter effect.

The explosion of video games outside the conventions of the AAA market has been a big help in keeping artistic talent in the business.

BRING DOWN THE HOUSE

>> Surprisingly, the rise of outsourcing has also provided a means for many veteran artists to stay with the industry when their families (or waistlines) can't handle the crunch-and-bust cycle anymore.

Freelancing or working at an outsourcing studio lets artists pick and choose projects that sync up with their lifestyles. Freelancing, outsourcing management, and consulting are all jobs in which experience and a large network of industry contacts offer an edge to the veteran.

Unfortunately, the flipside of the freelancer's freedom is economic insecurity and lack of benefits. Without the infrastructure of a studio setting, the life of a contractor or freelancer demands a lot of hustle, self-motivation, and risk tolerance, requirements that can be just as taxing as a conventional ship cycle. It's nice to be able to paint textures with your Wacom while rocking your baby to sleep with your free hand, but it's also important to do your homework before going solo. Suddenly becoming an entrepreneur when you have grown-up expenses and responsibilities is a serious challenge, one that demands more than just a great portfolio and a home copy of Maya.

To survive in a second career as a freelancer, you first need to make sure you have a solid

ILLUSTRATION BY DAVID HELLMAN

network of contacts who can really deliver work. You also need to know how to turn that work into contracts, which requires a solid understanding of deadlines, technical expectations, and review standards for your would-be partners before you make yourself dependent on them. Anybody who has ever lived as a freelancer for long knows all too well how easily verbal agreements, even between old friends and colleagues, can disguise enormous and potentially very expensive differences of opinion about what constitutes "done."

You'll also need to do some serious financial planning. Contract work is full of late payments, disputes about change orders, and promising projects that never materialize, so a substantial rainy day fund is very important. Successfully transitioning to life as an independent is a real challenge. The rewards (personal and, if you're lucky, financial) are great for those who can handle the inevitable downtime and the need for constantly seeking the next job.

If you're seriously considering going freelance, you should troll the GamingMercenaries.com forums for some frank discussion of ups and downs of the independent lifestyle.

BRING HOME THE BACON

>> One more important reason the industry has been seeing a few more gray hairs is money. Back in 2004, many artists in mid-career found they had run out of upside. With few directorial and management jobs, and no Hollywood-style seniority system, a lot of older artists found they could no longer expect much change in their compensation as they got older. Although the industry certainly has not followed the Hollywood or union model in terms of seniority, we have started paying veterans more, relatively speaking.

Receiving pay that's actually commensurate with experience has a lot to do with same trend toward bigger studios and more complex production that I alluded to earlier. The complexity of modern game development is just staggering. A big AAA game represents about the same amount of work as the complete design of a skyscraper—and that's the nuts and bolts, where-are-all-the-electrical-outlets design, not just the pretty concept sketches. The task is absurdly complex, and it only holds together because of the collective experience of the team.

In such a mind-numbingly tricky business, experience commands a price. In the most recent "Game Developer Salary Survey" (April 2009), line artists with more than six years of experience made about 50 percent more than entry-level artists, and that differential has been slowly increasing.

BRING IT ON

>> The flipside of experience, of course, is the danger of complacency. There's no question that our medium is constantly evolving; staying competitive is difficult. A regular production schedule and the demands of adulthood can make it hard to keep up with the latest tools and techniques. Plenty of veteran artists are traumatized when they see how their portfolio of professional work compares to the latest ambient-occluded, sub-surface-scattered 10,000,000-poly sculpture from some digital media arts grad. The hard truth is that the goalposts never stop moving.

For some of us, that's a positive. Plenty of grizzled vets still thrive on technological churn and find that the constant stimulus of new tools and tricks keeps the job from getting stale. Those who get tired of constantly relearning their jobs, on the other hand, face a tough set of choices.

Some gravitate toward jobs for which technological change is irrelevant, like concept art, management, or art direction. That transition can be rewarding and may also involve a welcome bump in perks and compensation, but all three of those areas are highly competitive and hard to break into. Some artists are able to migrate their existing skill sets to smaller platforms. The AAA world may not have much call for the ability to box-model a 500-poly character anymore, but the PSP, Nintendo DS, and mobile games still thrive on the parsimonious skills of the old school.

Most artists in mid-career fall somewhere between gung-ho technical enthusiasm and unrepentant technophobia. Many older artists can't spare the free time to troll through HighEnd3d.com every night and download every cool new normal-map ripper or Max Script that makes the rounds on the forums. At the same time, they're uneasily aware of the possibility that somebody, somewhere, may be making them obsolete. That, alas, is a feeling that even raw recruits learn pretty quickly. At least there are always high quality sources of information (say, magazine columns) to help keep you up on the latest developments.

BRING ME STRENGTH

>> Looking back five years, it seems that some aspects of life for the long-term game artist have definitely improved. The games business still makes heavy demands of all developers, but at least we're not driving out our most senior people with the same regularity we used to.

We're gradually coming to the point where we reward experience and create jobs that make good use of it. None of us has a clear idea of what a 15-, 20-, or 30-year career in games looks like (except perhaps Shigeru Miyamoto) but at least it's something we can begin to imagine. It's a start.

STEVE THEODORE *has been pushing pixels for more than a dozen years. His credits include MECH COMMANDER, HALF-LIFE, TEAM FORTRESS, and COUNTER-STRIKE. He's been a modeler, animator, and technical artist, as well as a frequent speaker at industry conferences. He's currently content-side technical director at Bungie Studios. Email him at stheodore@gdmag.com.*

BECOME A MASTER OF DIGITAL MEDIA APPLY NOW FOR 2009

We offer a 20-month master's degree in entertainment technology and digital media. This powerhouse graduate program combines industry-facing curriculum, real world projects, and a 4-month internship in Vancouver, Canada: videogame capital of the world.

Learn More. Drop by our booth at GDC Canada (Vancouver), contact alison_robb@gnwc.ca or visit **mdm.gnwc.ca**

DIGITAL KUNG-FU!

INTERACTIVE WORKSHOP | **MAY 14 2009**
AT VANCOUVER'S **CENTRE FOR DIGITAL MEDIA**

Get your digital black belt during a day of intensive training and professional development workshops. Register at **mdm.gnwc.ca/digitalkungfu**

CENTRE FOR
DIGITAL MEDIA

Masters of Digital Media Program
@ Great Northern Way Campus

UBC SFU emily carr university of art + design BCIT

a collaborative university campus environment

OUR CHEATIN' HEARTS

DESIGNING FAIRNESS IN GAMES

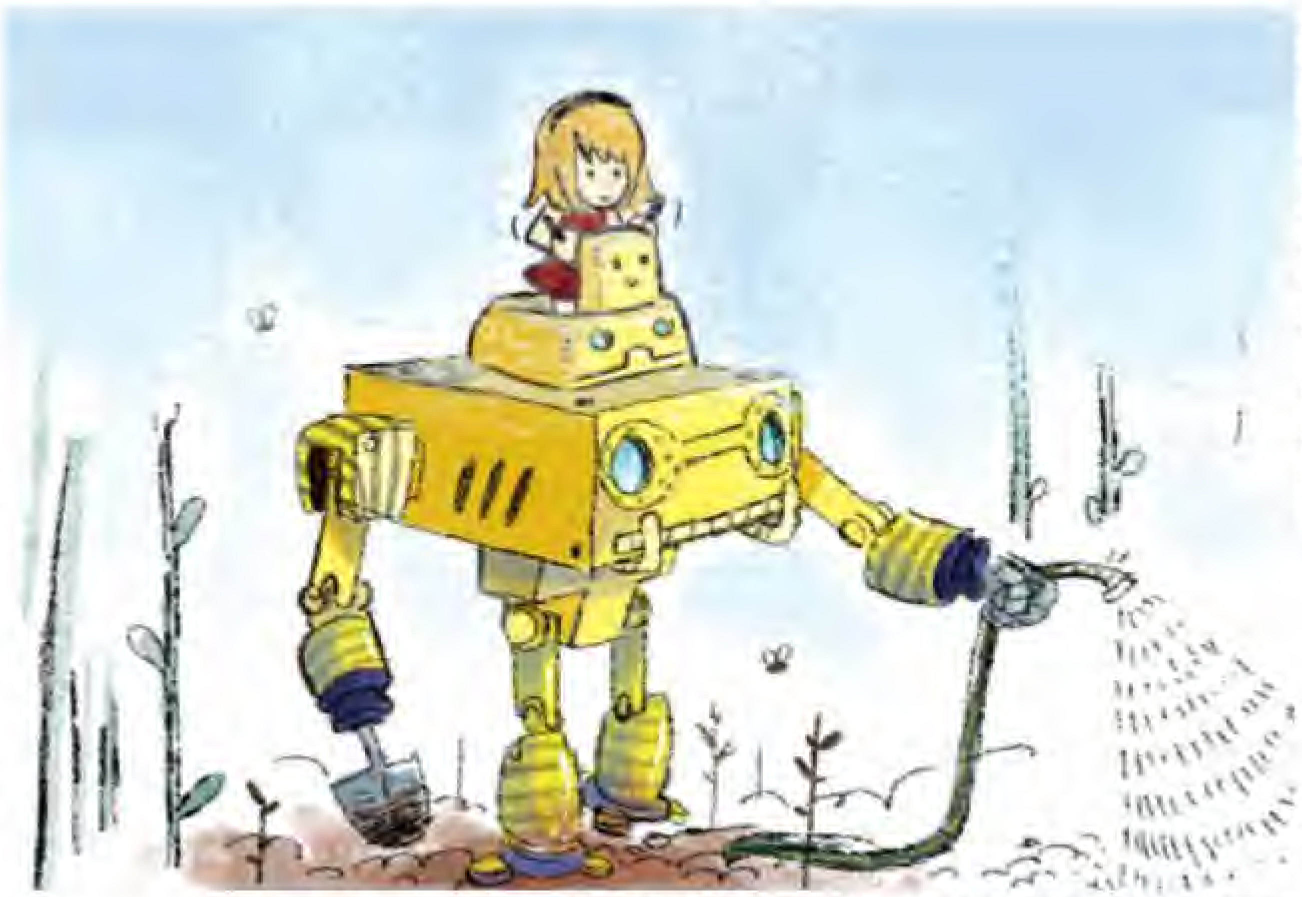


ILLUSTRATION BY MATT BRALY WITH JONATHAN KIM

Showing the mechanics that drive AI decisions helps alleviate player distrust.

THE DESIGNERS OF PUZZLE QUEST have a frustrating burden to bear. Everyone thinks they are a bunch of dirty cheaters.

The game includes a competitive version of BEJEWELED, in which players duel against the game's AI to create the most match-three sets. The problem comes from how the pieces on the game board are created. When three like-colored orbs line up, the player scores, the orbs are removed from play, and new pieces fall in to take their place. However, sometimes, these new pieces happen to be all the same color, which means that a new match is automatically made, and the player effortlessly scores again.

The odds of getting this result are low (around two percent), but it happens often enough that a player will see it many times with enough games played. The problem is that the AI is playing the same game, and the player sees the same good luck fall into the enemy's lap from time to time, too.

At this point, human psychology takes over. Because the new pieces are hidden from view, how does the player know that the computer is not conducting some funny business and giving itself free matches?

The human mind is notoriously bad at grasping probability, leaving many players convinced that the AI is cheating. The developers at Infinite Interactive have pledged over and over again that the code operates fairly, but whether they like it or not, the player's experience is affected by the possibility of unfairness.

TRUST ME

>> Players don't trust games from the outset. Trust needs to be earned over time. Our audience is well aware that we developers can make a game do whatever we want under the hood, so the transparency and consistency of a game's rules contribute significantly to player immersion.

The worst feeling for players is when they perceive—or more accurately, suspect—that a game is breaking its own rules and treating the human unfairly.

This situation is especially challenging for designers of symmetrical games, in which the AI tries to solve the same problems as the human. On the other hand, for asymmetrical games, cheating is simply bad game design. Imagine the frustration that would result from enemies in HALF-LIFE warping around the map to flank the player, or the guards in THIEF instantly spotting a player hiding in the shadows.

However, under symmetrical conditions, the AI often needs to cheat just to be able to compete with the player. Accordingly, designers must learn which manifestations of AI cheating feel fair to a player and which do not. As the PUZZLE QUEST team knows, games need to avoid situations in which players suspect that the game is cheating them.

What do we really mean when we talk about whether the AI is cheating? It's not the same as increasing the difficulty level, in which case players expect the game to provide extra challenges for them. Rather, cheating is a matter of unfairness: Is the player rewarded for successful plays and not arbitrarily punished just to maintain the challenge?

Unfortunately, in practice, the distinction between difficulty levels and cheating is not so clear.

SHOW THE MECHANICS

>> Fans of racing games are quite familiar with this gray area. A common tactic employed by AI programmers to provide an appropriate level of challenge is to "rubber band" the cars together. In other words, the code ensures that if the AI cars fall too far behind the human, they automatically speed up. On the other hand, if the human falls behind, the AI slows down while the player catches up.

Rubber banding is often obvious to players, which can dull their sense of accomplishment when they win and raises suspicions when they lose. Ironically, games that turn rubber banding into an explicit game mechanic often become more palatable to players. The MARIO KART series, for example, has a history of disproportionately divvying out rewards via mystery boxes relative to each driver's current standing. While the first-place racer might receive a shell (only useful for attacking other cars in the lead), players in the rear might get a speeding bullet, which automatically warps them to the middle of the pack.

These self-balancing mechanics aren't unique to electronic games; in *Settlers of Catan*, for example, the robber can block the leader's tiles. These explicit mechanics don't seem like cheating because the game is so explicit about how the system works. Thus, players understand that the bonuses available to the AI will also be available to them if they fall behind. With cheating, perception becomes reality, so transparency is the antidote to suspicion and distrust.

CHEATING IN CIVILIZATION

>> Sometimes, hidden bonuses and cheats are necessary to provide the right challenge for the player. The CIVILIZATION series provides plenty of examples of how this process can go awry and drive players crazy with poorly handled cheating.

Being turn-based, the developers could not rely on a human's natural limitations within a real-time environment. Instead, CIVILIZATION gives out a progressive series of unit, building, and technology discounts for the AI as the levels increase (as well as penalties at the lowest levels). Because of their incremental nature, these advantages for the AI have never earned much ire from players. Their effect is too small to notice on a turn-by-turn basis, and players who pry into the details usually understand why these bonuses are necessary.

On the other hand, many other inequalities between the player and the AI have struck the audience as being unfair. In the original version of the game, the AI could create units for free under the fog-of-war, a situation that clearly showed how the computer was playing by different rules than the human. Also, AI civilizations would occasionally receive free "instant" Wonders, often robbing a player of many turns of work. While an AI beating the human to a Wonder using the slow drip of steady bonuses was acceptable, granting it the Wonder instantly felt entirely different.

How a cheat will be perceived has much more to do with the inconsistencies and irrationality of human psychology than any attempt to measure up to some objective standard of fairness. Indeed, while subtle gameplay bonuses might not bother a player, other legitimate strategies could drive players crazy, even if they know that a fellow human might pursue the exact same path as the AI.

In the original CIVILIZATION, the AI was hardwired to declare war if the player was leading the game by 1900 AD. This strategy felt unfair to players, who believed that the AI was ganging up on them, even though most would have followed the same strategy without a second thought in a multi-player game.

In response, by the time Firaxis rolled out CIVILIZATION III, we guaranteed that the AI did not consider whether an opponent was controlled by a human or a computer when conducting diplomacy. Still, these changes did not inoculate the dev team against charges of unfairness. CIVILIZATION III allowed open trading, letting players swap technology for maps or resources for gold. Enterprising players would learn when to demand full price for their technologies and when to take whatever they could get—from a weak opponent with very little wealth, for example.

We adapted the AI to follow this same tactic so that it would be able to take whatever gold it could from a backward neighbor. To the players, though, the AI appeared to be once again ganging up against them. Because the AI civilizations were fairly liberal with trading, they tended to be around the same technology level, which led the player to believe that they were forming their own non-human trading cartel, spreading technologies around like candy (or, in the parlance of the forums, "tech-whoring").

Infinite Interactive's PUZZLE QUEST: CHALLENGE OF THE WARLORDS

PERCEPTION IS REALITY

>> Once again, perception is reality. The question is not whether the AI is playing "fairly," but what experience it creates for the player.

If questions of fairness keep creeping into the player's mind, the game needs to be changed. Thus, for CIVILIZATION IV, we intentionally crippled the AI's ability to trade with one another to ensure that a similar situation did not develop.

The computer is still a black box to players. Single events based on hidden mechanics need to be handled with great care. Developers of sports games, for example, need to be very sensitive to how often a random event hurts the player, such as a fumble, steal, or ill-timed error. The dangers of perceived unfairness are simply too great.

Returning to our original example, the developers of PUZZLE QUEST actually should have considered cheating—but in favor of the player. The game code could ensure that fortunate drops only happen for the human and never for the AI. The ultimate balance of the game could still be maintained by tweaking the power of the AI's equipment and spells, changes that appear fair because they are explained explicitly to the player. The overall experience would thus be improved by the removal of these negative outliers that only serve to stir up suspicion.

When the question is one of fairness, the player is always right.

SOREN JOHNSON *is a designer/programmer at EA Maxis, working on an unannounced project. He was the lead designer of CIVILIZATION IV and the co-designer of CIVILIZATION III. Read more of his thoughts on game design at www.designer-notes.com. Email him at sjohnson@gdmag.com.*

resources

Falstein, Noah. "Fair Play," Game Shui, *Game Developer*, August 2006.

West, Mick. "Intelligent Mistakes: How to Incorporate Stupidity Into Your AI Code," *Game Developer*, April 2008.

West, Mick. "Texas Hold'em AI," The Inner Product, *Game Developer*, November 2005.

THE AUDIO DESIGN OVERVIEW

NOTING THE ESSENTIALS

IN "WORKING IN CONCERT" (JANUARY 2009), I EXAMINED THE NEEDS OF different parts of the development team regarding audio documentation on a project. Once published, I received a slew of emails asking for copies of sample documents to help further illustrate the points I had discussed. Unfortunately, as with nearly everything in our industry, those documents are full of proprietary information, and thus confidential.

However, I am going to share here some information about what goes into a standard audio design overview document. This multipurpose document can help communicate a game's overarching audio plan to producers, composers, and sound designers while simultaneously helping you organize your aural plans for a game into a cohesive structure.

PURPOSE AND STRUCTURE

» At a macro level, an audio design overview will contain a minimum of four sections: an introduction followed by separate sections for sound effects, music, and voice. Even if your game doesn't contain all of those disciplines, incorporate sections for each and simply include a statement saying, "[Game] will not include any voice."

The introduction spells out to the reader what the document is, what kind of information it contains, and a high-level summary of what else is in the document. Introductions may contain a section detailing the various authors of the audio design overview and what their roles will be on the project.

Another section that's very common in the introduction is one that outlines the goals of the audio design overview. As with all information in an overview document, these goals can either be written as paragraphs or listed as bullet points. For example:

> C. GOALS
> The goals of this document are:
> - To define the scope of the work and its pipeline,
> - To schedule the work against the deadlines of the project, and
> - To define any outstanding questions so that we can pursue resolution.

Remember that the introduction is a summary, not a place to get bogged down in details. Just outline for the readers what they should expect to get out of reading the document.

DISCIPLINE DETAILS

» The sections outlining each of the disciplines can come in any order, though perhaps most common is: sound effects, music, voice. Each section should be broken down into at least two subsections: aesthetic style guide and technical requirements.

The purpose of the style guide is to explain how the aural approach for the game falls in line with the overall aesthetics of the project. Is it a sports game meant to capture the feel of watching *Monday Night Football*? If so, explain the plans for helmet-crunching foley or the network TV sports UI effects that will populate the HUD. If the game is a space shooter, talk about how realistic or hyper-realistic the audio will be—is it more like DEAD SPACE or *Duck Dodgers*? Give examples from television, movies, and games that approximate your aesthetic goals.

When talking music aesthetics, mention genres of music, orchestral versus pop/rock instruments, and whether you want wall-to-wall music or a more nuanced interplay between sound effects, ambiences, and soundtrack. In the voice section, talk about casting. If you have kids in your game, do you cast adults who sound young or child actors? Are you going to cast locally, or is there a benefit to casting actors across the globe? Is there a fully voiced script that needs to be localized, or does the voice work consist of only iconic vocal utterances?

The technical sections for each discipline can vary in their depth of detail. If you're planning to create separate documents that outline audio technical requirements in greater detail, you might want to use the audio design overview to give just a summary of the audio tech needs. If, on the other hand, the document is going to serve as the bulk of your documentation, you'll want to be very specific about the tech requirements.

For sound effects, spell out your needs for streaming, stream management, sound prioritization, and the loading and management of resident sounds. Talk about needs like DSP, pitch, pan, and volume variation, or zone-based ambience implementation and occlusion.

For music, discuss how many streams will be needed, whether the music is interactive or dynamic, fade or crossfade needs, or any tech that isn't inherent to the platform, like customizable soundtracks.

For voice, have subsections on anything from concatenated stitching, to ducking, to hit react voice interrupt systems. If you're planning to use middleware like FMOD or Wwise, discuss their uses in these sections.

SUPPORTING SPECIFICS

» Staffing plans, production pipeline maps, and specific technical needs documents can all be supplemental supports to the audio design overview. If, however, you chose to only have the overview document, you may want to consider including more sections than the four spelled out here.

If there will be cut scenes in the game, consider a section that deals with their tech, pipeline, and aesthetic needs. If you'll be outsourcing some of the production work, talk about hiring and management plans. If you're going to be releasing on multiple SKUs, spell out what the different approaches will be for each SKU so as to best ensure a cohesive audio experience from one platform to the next.

Comprehensive, cohesive documentation will help ensure that the audio team and the development team both understand their roles and expectations.

JESSE HARLIN *has been composing music for games since 1999. He is currently the staff composer for LucasArts. You can email him at jharlin@gdmag.com.*

PlayStation.

We are HIRING across all disciplines:
programmers, artists, producers, designers, etc.

Apply online: http://www.us.playstation.com/Jobs

SONY
COMPUTER ENTERTAINMENT

BLIZZARD®
ENTERTAINMENT

IS HIRING

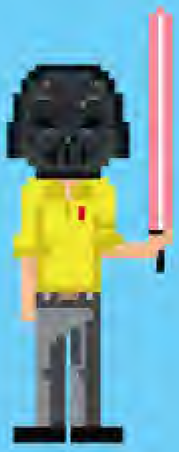
We are actively recruiting across all disciplines for the following locations:

Irvine, California | Austin, Texas | Velizy, France | Cork, Ireland
Seoul, South Korea | Shanghai, China | Taipei, Taiwan

www.blizzard.com/jobs

© 2009 Blizzard Entertainment, Inc. All rights reserved.

{newbie}



{alpha geek}

Please geek responsibly.

You may speak the language,
but are you geeked?
Here's a chance to prove it.

LEARN:

ADVANCING COMPUTER SCIENCE	NETWORK ENGINEERING
ARTIFICIAL LIFE PROGRAMMING	NETWORK SECURITY
DIGITAL MEDIA	ROBOTICS & EMBEDDED SYSTEMS
DIGITAL VIDEO	TECHNOLOGY FORENSICS
GAME ART AND ANIMATION	TECHNOLOGY MANAGEMENT
GAME DESIGN	VIRTUAL MODELING & DESIGN
GAME PROGRAMMING	WEB & SOCIAL MEDIA TECHNOLOGIES

University of Advancing Technology
UAT
Learn. Experience. Innovate.

www.uat.edu > 877.UAT.GEEK
877.828.4335

ACADEMY *of* ART UNIVERSITY

FOUNDED IN SAN FRANCISCO 1929 BY ARTISTS FOR ARTISTS

Artwork by Jaehoon Kim, Animation & Visual Effects Student

Artwork by Sungjae Cho, Animation & Visual Effects Student

Artwork by Diego S. Velasquez, Animation & Visual Effects Student

Advertising
Animation & Visual Effects
Architecture (M-ARCH)
Computer Arts New Media
Fashion
Fine Arts
Graphic Design
Illustration
Industrial Design
Interior Architecture & Design
Motion Pictures & Television
Multimedia Communications (BA, MA)
Photography

ENROLL NOW
STUDY ONLINE OR IN SAN FRANCISCO

- Degree Programs
- Continuing Art Education Courses
- High School Programs

800.544.2787
WWW.ACADEMYART.EDU

79 NEW MONTGOMERY STREET,
SAN FRANCISCO, CALIFORNIA 94105

Accredited member WASC, NASAD, Council for
Interior Design Accreditation (BFA-IAD), NAAB (M-ARCH)

Create the Game

Gaming Degree Programs for the Next Generation

Game Art
Bachelor's Degree Program

Game Development
Bachelor's Degree Program

Game Design
Master's Degree Program

ANIMATION | DESIGN | ENTERTAINMENT BUSINESS | FILM | RECORDING ARTS | SHOW PRODUCTION | VIDEO GAMES | WEB

Master's | Bachelor's | Associate's Degrees

800.226.7625 • 3300 University Boulevard • Winter Park, FL 32792
Financial aid available to those who qualify • Career development assistance • Accredited University, ACCSCT

fullsail.edu *Online Programs Available*

FULL SAIL
UNIVERSITY.

June 2-4, 2009 • L.A. Convention Center • E3Expo.com

| Disney Interactive | Konami
| Nintendo of America Inc. | Atari, Inc. | Logic
Square Enix, Inc. | So k Games | TOMY | Sonic Ga
Namco Bandai Games A thesda Softworks | Di
Natsume | Game Source | C nent | E Games for Less
KOEI Corporation | Playsea os Interactive | CH Products
Ubisoft Entertainment, Inc. | Br es | Dreamcatcher Games | SEGA
Capcom USA, Inc. | Griffin Inter ma Games/Random House, I
V Games | dreamGEAR, LLC | ony Computer Entertainme
igital Entertainment | KOEI Corp nepark Holdings | Eidos In
THQ, Inc. | Gametech | IGN Ente yko Technologies | MTV
Microsoft Corporation | al River | Hi Rez Studios
Majesco Entertainme avo LTD | Epic Gam
Products | Electronic Arts | Take-Two Interactive | U
Gamers First | Activision | Brady Games
n | Warner Bros. Interac

E3 expo

International Media Sponsor
MCV

Official Show Daily Sponsor
GAMEPRO MEDIA

E3 Expo is a trade event and only qualified industry professionals may attend. No one under 17 will be admitted, including infants. Visit www.E3Expo.com for registration guidelines.

© Entertainment Software Association 2009

One Time. One Place. One Event.

See the biggest games and the best names in interactive entertainment.
They're all in one place, at one time, for the one event of the year.
Check out the full exhibitor list at www.E3Expo.com. Register today!

esa entertainment software association

Produced by IDG WORLD EXPO

CALIFORNIA DEVELOPERS

Connect with the Largest Educational System in the World

110 Colleges, 2.6 million students, affordable, customized training

MEI Game & Simulation
SPECIAL INTEREST GROUP

meigamesim.intronetworks.com

Train Your Employees
Apply for State and Federal Grant Programs
Access Business Development Resources

California Community Colleges

ECONOMIC & WORKFORCE DEVELOPMENT

Multimedia & Entertainment Initiative

MAKE MORE ENEMIES

Game Design at Vancouver Film School shows students how to make more enemies, better heroes, cooler levels, and tighter connections to the industry.

In just one year, you'll learn every aspect of game design. Your portfolio project is a playable video game.

VFS grads get snapped up by top companies like BioWare, Radical, Relic, and Ubisoft, and the *LA Times* named VFS a top 10 school "most favored by video game industry recruiters".

VFS

vfs.com/enemies

VFS student work by **Thaddeus Maharaj**

CENTER FOR DIGITAL IMAGING ARTS AT BOSTON UNIVERSITY

3D ANIMATION
+INTERACTIVE MEDIA

GAME ART+CHARACTER ANIMATION

TWO CAMPUS LOCATIONS
WALTHAM, MA & WASHINGTON, DC

Financial assistance and career services available.
Now accepting applications. *Apply today!*

cdia
BOSTON UNIVERSITY
CDIABU.COM

Make Your World Your Career

Today's world is all about podcasts, video games, and Facebook.®

Take your passion to the next level with a bachelor's degree in Digital Entertainment and Interactive Arts with a concentration in Video Game Art and Animation from National University

Learn more today at
Getinfo.nu.edu/Game
or **800.NAT.UNIV**
628.8648

The University of Values

San Jose Campus:
3031 Tisch Way
100 Plaza East
408.236.1100

© National University 2009.

ADVERTISER INDEX

COMPANY NAME	PAGE	COMPANY NAME	PAGE
Academy of Art University	52	Perforce Software	30
Activision	19	Pocket Soft	41
Blizzard Entertainment	50	Rad Game Tools	C4
Center for Digital Imaging	55	Santa Barbara City College	54
Epic Games	14	Seapine Software	6
Eyetronics	C2	Sony Computer Entertainment	49
Full Sail Real World Education	53	The National University System	55
Havok	10	Trinigy	3
Intel	31–35, C3	University of Advancing Technology	51
Masters of Digital Media	45	Vancouver Film School	54
Neversoft	23	Xaitment	13
Nokia	42		

Game Developer (ISSN 1073-922X) is published monthly by United Business Media LLC, 600 Harrison St., 6th Fl., San Francisco, CA 94107, (415) 947-6000. Please direct advertising and editorial inquiries to this address. Canadian Registered for GST as United Business Media LLC, GST No. R13288078, Customer No. 2116057, Agreement No. 40011901. **SUBSCRIPTION RATES:** Subscription rate for the U.S. is $49.95 for twelve issues. Countries outside the U.S. must be prepaid in U.S. funds drawn on a U.S. bank or via credit card. Canada/Mexico: $69.95; all other countries: $99.95 (issues shipped via air delivery). Periodical postage paid at San Francisco, CA and additional mailing offices. **POSTMASTER:** Send address changes to Game Developer, P.O. Box 1274, Skokie, IL 60076-8274. **CUSTOMER SERVICE:** For subscription orders and changes of address, call toll-free in the U.S. (800) 250-2429 or fax (847) 647-5972. All other countries call (1) (847) 647-5928 or fax (1) (847) 647-5972. Send payments to *Game Developer*, P.O. Box 1274, Skokie, IL 60076-8274. For back issues write to *Game Developer*, 4601 W. 6th St. Suite B, Lawrence, KS 66049. Call toll-free in the U.S./Canada (800) 444-4881 or fax (785) 838-7566. All other countries call (1) (785) 841-1631 or fax (1) (785) 841-2624. Please remember to indicate *Game Developer* on any correspondence. All content, copyright *Game Developer* magazine/United Business Media LLC, unless otherwise indicated. Don't steal any of it.

GETTING TO NO

THE MOST EFFECTIVE WAYS TO DEFLECT WORK

HOW MANY OF US HAVE HAD THE EXPERIENCE OF HAVING A LEISURELY DAY at the office of web surfing and forum participation rudely interrupted by somebody who wants us to actually do something? Sure, maybe there's some fantastic new feature for the game that someone is proposing. But that will mean work for you! Here's how to put a stop to this kind of thinking before it turns real.

BIG SIGH! When you hear someone potentially creating work for you, look immediately to the first weapon in your arsenal—the BIG SIGH. Sigh as deeply as you can and let your eyes roll upwards a little (not too much, or it will seem unnatural). The BIG SIGH establishes right off the bat how you're going to approach this, especially when you use it with nuance. For example, it could be a condescending sigh that says, "I can't believe you're going to bother me about this," or it could be a passive-aggressive sigh designed to evoke pity, like, "Here I am, buried under an impossible amount of work, and now you're going to add some. Well, if there's one thing I'm used to by now it's suffering, so I guess I'll deal with it. Or I won't, and keel over dead."

"That's not in the schedule." Your second line of defense is to invoke the schedule. Never mind that you don't pay attention to the schedule yourself. Use it as a weapon when it suits you! If somebody's got a proposal you don't like, be sure to point out how it will negatively impact the game's ability to ship on time. For bonus points, say this within earshot of a producer—you might get on their good side!

"Cool idea, but it's a bit late in the project cycle to implement now. Next game!" It's pretty remarkable how many games in a row you can keep up the whole "next game" ruse. By the time you reach the fourth or fifth go around this loop, the requesting party is probably on their way out the door. That is, if the studio is still in business.

"That's totally unproven in the market. The sales data proves it." If raising the specter of actually shipping the title doesn't throw them off their dogged path, it's time to start picking on the idea itself. As Herman Melville once said, "It is better to fail in originality than succeed in imitation." And Melville died poor and unknown, so why would anyone want to do that? Risk is the bane of the industry, especially when millions of dollars are on the line. So point out that the investment is at risk when anything that deviates from the norm is proposed. You can count on the people who control the purse strings to see it your way.

"Yeah, well, we're not Valve." Someone's got a great idea inspired by the smart and well-respected developers at Valve Software. But hey, if you had unlimited time and money for your game project, you'd all just be doing whatever you wanted, right? Such as working at Valve. The point is, most projects must acknowledge they have hard limits on time and money, and if you can tie whatever it is you don't want to do to those limits, you're pretty much set. Plus, doing this lets you shrug your shoulders with indifference—you don't have to be the bad guy, it's the whole *world* that just isn't fair! Just keep saying "we're not Valve," and before you know it, it'll be a team-wide mantra.

ILLUSTRATION BY DAN PALADIN

"That's the stupidest thing I ever heard." When all of your picking on the idea fails, try to frighten them with what appears to be your massively superior intellect. Angrily say the idea is unworkable because of some incredibly obvious reason or other, and that you can't believe they didn't realize it immediately like you did. Since nobody wants to be seen as dumb, they'll probably back down right away instead of calling your bluff. It doesn't really matter what the problem you point out actually is—it could be UV mapping, or endianness, or something related to the Pauli exclusion principle.

"As much as I love your idea, I just don't think that's what the rest of the team wants ... " Ah, the ferment of studio politics! This is the Machiavellian power play you can try if you're the heartless type. Pit the requester against the rest of the team. He'll start to feel weak and isolated. He might lash out. Make him think you're on his side. "I know it's unfortunate, but I'm really the only one here who seems to recognize your talents." Playing Puppet Master is a lot of work. But it's probably more entertaining than whatever it was the guy was about to ask you to do.

"Seems all right to me, but please run that through the appropriate channels." This is sort of a stalling technique if none of the other things you've tried have worked. You can only hope that the person who's making the request doesn't even know what the proper channels are—which is pretty likely in a game studio. But there's always the danger that against all odds, the request will somehow get all the way through, and—of all the indignities— you'll actually have to do the work. That doesn't mean you have to be happy about it, though. Start prepping another BIG SIGH.

MATTHEW WASTELAND *is a pseudonymous game developer who has a fairly common first name. Email him at* ***mwasteland@gdmag.com.***

Tools for mortals to achieve the divine

When crafting epic battles of the nearly divine, a developer's weapon of choice must have both power and precision, optimizing performance and engaging the senses in real time. The Intel® Graphics Performance Analyzers Suite opens new doors in game performance and expands the horizon to new customers.

"PC developers have never had such a reliable, flexible, and extensive tool set for identifying and resolving complex graphics performance issues."

— **BARTOSZ KIJANKA,** GAS POWERED GAMES

Get more information on Intel Graphics Performance Analyzers:
www.intel.com/software/gpa

To get the latest on software development for visual computing, visit:
www.intel.com/software/visualadrenaline

VISUAL ADRENALINE intel Software

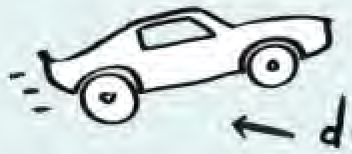
Copyright 2009 Gas Powered Games Corp. All rights reserved.
Gas Powered Games is the exclusive trademark of Gas Powered Games Corp.

THERE ARE LOTS OF WAYS TO BE AN EVEN

COOLER

GAME DEVELOPER

like grow a mullet



drive a sweet old Camaro

or you can use BINK VIDEO.

You get an amazing, super FAST video and audio codec - all in a simple, clean API. And Bink Video runs on EVERY PLATFORM!

BINK VIDEO

USING BINK VIDEO NOT ONLY MAKES YOU cooler, IT MAKES YOU rad!

RAD GAME TOOLS ©

www.radgametools.com
(425) 893-4300